Universidad Carlos III de Madrid
www.uc3m.es

TRABAJO DE FIN DE GRADO

DESARROLLO DE APLICACIÓN DE SEGURIDAD VIAL EN ANDROID

Autor

Jesús Urdiales de la Parra

Grado en Ingeniería Electrónica Industrial y Automática

Tutor

Fernando García Fernández

FEBRERO DE 2016

UNIVERSIDAD CARLOS III DE MADRID

Departamento de Ingeniería de Sistemas y Automática

Agradecimientos

A mis padres y a mi familia por haberme apoyado tanto durante estos años de estudio. A mis amigos por haberme hecho pasar tan buenos momentos. A mis profesores y tutores por haberme formado hasta llegar a ser quien soy.

ÍNDICE GENERAL

Índice de Abreviaturas y acrónimos

OpenCV: open source computer vision.

OMS: organización mundial de la salud.

ADAS: Advanced Driver Asistance Systems o sistemas avanzados de ayuda a la conducción.

fps: frames per second o imágenes por segundo.

SO: sistema operativo, en inglés OS (Operative System).

IDE: Integrated Development Environment o entorno de desarrollo integrado.

ADT: Android Development Tools o herramientas de desarrollo para Android.

TFG: Trabajo de fin de grado.

JNI: Java Native Interface o Interfaz nativa de Java.

GUI: Graphical User Interface o Interfaz gráfica de usuario.

NDK: Native Development Kit o herramientas de desarrollo nativo.

SDK: Software Development Kit o herramientas de desarrollo de software.

LSI: Laboratorio de Sistemas Inteligentes.

IVVI: Intelligent Vehicle based on Visual Information o vehículo inteligente basado en información visual.

HOG: Histogram of Oriented Gradients o histograma de gradientes orientados.

XML: eXtensible Markup Language o lenguaje de marcas extensible.

UKF: Unscented Kalman Filter.

Índice de figuras

Índice de tablas

RESUMEN

El continuo desarrollo de la tecnología aplicada a los dispositivos móviles inteligentes (smartphones en inglés) ha hecho posible la comercialización de dispositivos cada vez más potentes a nivel de procesamiento de la información. Tanto es así que hoy en día son capaces incluso de ejecutar con soltura aplicaciones basadas en el procesamiento de imágenes y en realidad aumentada.

Otro campo en continuo desarrollo es el de los ADAS o sistemas avanzados de asistencia a la conducción. En este ámbito de la industria del transporte por carretera se desarrollan todo tipo de aplicaciones, vehículos o sistemas de ayuda encaminados a facilitar la conducción y mejorar la eficiencia y la seguridad de la misma.

Este proyecto une la capacidad de procesamiento de los smartphones con el campo de los ADAS para desarrollar una aplicación en el entorno Android capaz de detectar peatones, realizar un seguimiento de los mismos y estimar la distancia a la que se encuentran utilizando únicamente una sola cámara. Posteriormente se han añadido estas mismas capacidades aplicadas en la detección de coches.

La etapa de detección de peatones o coches está basada en los detectores Haar Cascade y HOG presentes en las librerías de visión por computador OpenCV. La estimación de distancias hace uso del modelo Pin Hole para poder realizar dicha estimación utilizando un sistema monocular. Finalmente la etapa de “tracking” o seguimiento opera sobre un filtro de Kalman.

La aplicación está diseñada para funcionar colocando el dispositivo, smartphone o tableta, sobre el cristal delantero del coche, del mismo modo que un GPS. Se trata por tanto de un sistema asequible, ya que requiere únicamente de un smartphone, algo que gran parte de la población posee y utiliza en su vida diaria. Además es totalmente portable y aplicable para todo tipo de vehículos terrestres.

Al final de esta memoria se muestran los resultados obtenidos tras realizar varias pruebas a la aplicación terminada. Además se analizarán posibles formas de mejorar la aplicación y distintas posibilidades de ampliación de su funcionalidad para su desarrollo en futuros proyectos.

ABSTRACT

The development of technology in recent years has made possible the commercialization of increasingly powerful smartphones. These devices are even able to run computer vision and augmented reality based applications.

Another area of industry in continuous development is the ADAS (advanced driver assistance systems). This field of the road transport industry develops applications, vehicles and support systems whose aim is to make driving easier and improve the efficiency and safety of the same.

In this project I developed an application for Android that combines the processing power of smartphones with the field of ADAS. The application is able to detect pedestrians, track them and to estimate the distance they are using only one camera. Later I added these same capabilities applied in detecting cars.

The step of detecting pedestrians or cars is based on the Haar Cascade and HOG detectors present in the computer vision library OpenCV. Distance estimation makes use of the Pin Hole model to make this estimate using a monocular system. Finally the stage of tracking operates on a Kalman Filter.

The application is designed to work by placing the device, smartphone or tablet, on the windscreen of the car, just like a GPS. It is therefore an affordable system, requiring only a smartphone, which owns most of the population and use in their daily lives. It is also totally portable and applicable to all types of land vehicles.

At the end of this memory the results obtained after several tests of the finished application are shown. Moreover possible ways to improve the application are analyzed. I also discuss possibilities of expanding its functionality in future projects.

1. INTRODUCCIÓN

1.1. Motivación

Hoy en día el sector del transporte tiene un papel fundamental en el desarrollo de cualquier sociedad y actividad. En concreto el transporte terrestre cuenta actualmente con un parque de 775 millones de vehículos, cifra que aumenta año tras año, en parte debido al desarrollo del mismo en países emergentes como la India y China. Se prevé que para 2020 el parque mundial pase a ser de 1000 millones, llegando a los 2000 millones en 2050.

Sin embargo, a pesar de su importancia en la sociedad, este continuo crecimiento del sector tiene una serie de consecuencias negativas para la población. Estas son la congestión del tráfico que tal cantidad de vehículos provoca, el elevado número de accidentes y problemas de contaminación y salud.

La congestión del tráfico es un problema que se concentra principalmente en entornos urbanos, y se ha podido comprobar que la técnica clásica de aumentar el número de carriles de las vías no es una solución. En cuanto al problema de los accidentes, el transporte terrestre constituye una de las principales causas de muerte a nivel mundial. La Organización Mundial de la Salud [1] estima que el transporte por carretera ocasiona 1,25 millones de muertos y 10 millones de heridos al año. Además, en sus informes, la OMS asegura que los accidentes por carretera son la primera causa de muerte en jóvenes de entre 15 y 29 años, y que 3 de cada 4 fallecidos son hombres. En la imagen inferior se puede observar como la probabilidad de morir en un accidente de carreta depende enormemente del lugar de residencia y que el 22% de los fallecidos son peatones.

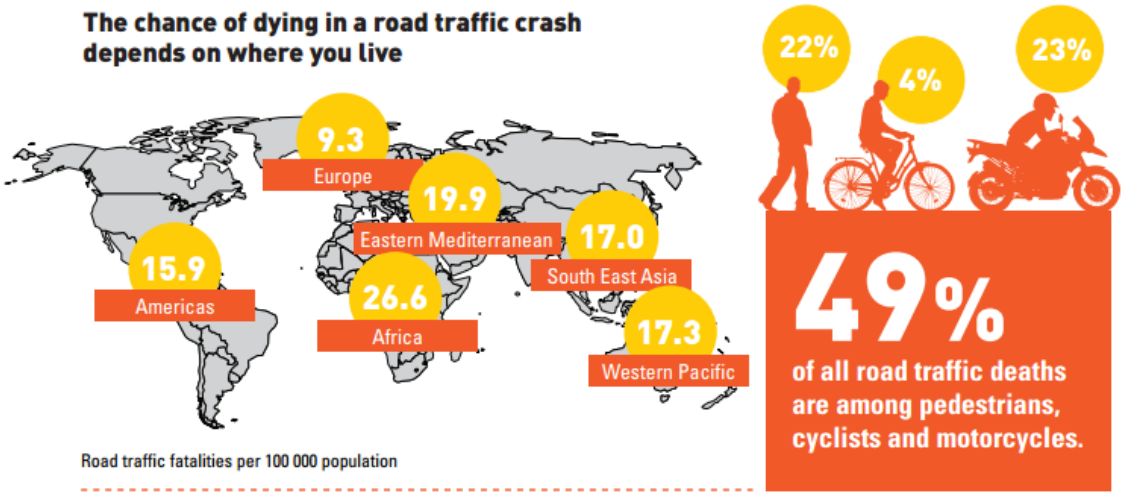


Figura 1: extracto del informe de la OMS sobre seguridad vial [1]

Por último, la contaminación causada por la combustión de los automóviles está contribuyendo al efecto invernadero, lo que nos lleva a un aumento del cambio climático. Además, en los últimos años se ha podido comprobar como esta contaminación incrementa los problemas respiratorios a la vez que disminuye la esperanza de vida de la población.

A todos los problemas anteriores hay que sumar el enorme impacto económico que todos ellos provocan. La solución más prometedora consiste en impulsar el desarrollo de sistemas avanzados de asistencia a la conducción, hasta llegar a los coches autónomos. Entendemos por coche autónomo aquel que es capaz de realizar todas las tareas necesarias para moverse de un punto a otro sin ninguna intervención humana.

El uso de sistemas avanzados de asistencia a la conducción (ADAS) y posteriormente del coche autónomo, no solo podrá resolver todos los inconvenientes anteriormente descritos, sino que además traerá otros beneficios tales como un mayor aprovechamiento del tiempo mientras se conduce, el uso sin limitaciones por edad o discapacidad del coche o la desaparición de las multas de tráfico.

A continuación se presenta una imagen donde se muestra el futuro del coche autónomo de aquí al año 2050, según un informe de Morgan & Stanley [2] redactado en el año 2014. Como se puede observar, hoy en día los vehículos no son capaces aún de conducir por sí solos, sino que los sistemas avanzados de asistencia a la conducción toman el control única y exclusivamente cuando el coche prevé un accidente inminente. Se espera que para este año, 2016, existan ya vehículos capaces de tomar el control total de la conducción si el conductor así lo desea. Para 2019 se espera que los coches sean totalmente autónomos, aunque en el asiento del conductor siempre habrá alguna persona para actuar en caso de emergencia. Por último, el informe predice que para 2022 ya se puedan ver los primeros coches en circulación totalmente autónomos, sin conductor. Finalmente se espera que para 2030 el 30 % de los vehículos de pasajeros circulen de esta forma.

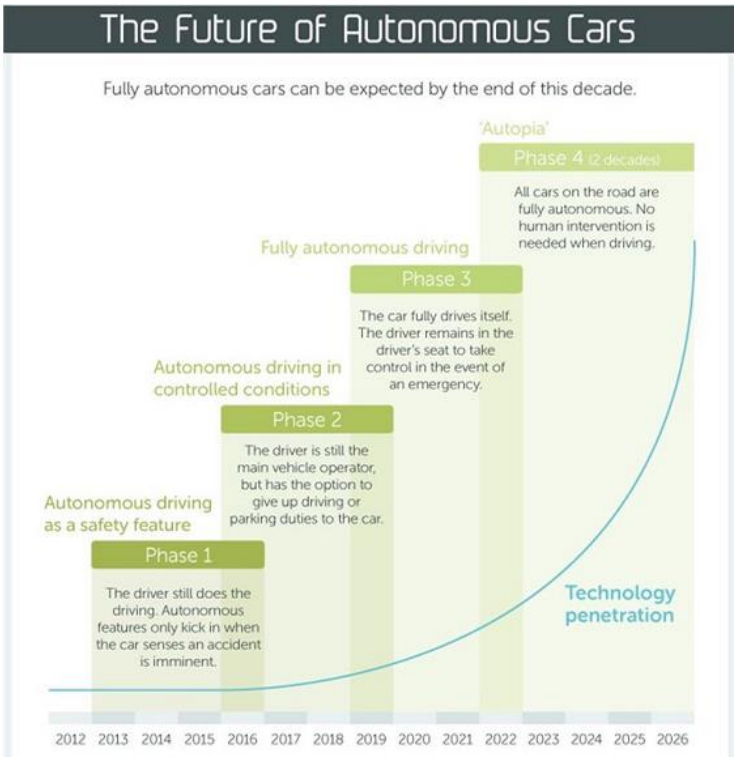


Figura 2: Futuro de los coches autónomos [2]

Grandes empresas del sector del automóvil llevan años desarrollando sistemas y tecnologías en el campo de los ADAS. Entre ellas tenemos el control de crucero inteligente, el aviso por

cambio involuntario de carril, los sistemas de vigilancia de ángulo muerto, sistemas de parada automática en caso de emergencia, sistemas de detección de peatones a través de cámaras monoculares, estéreo e infrarrojas, entre otros muchos.

Sin embargo, todas estas tecnologías se desarrollan y utilizan en muchos casos en un determinado vehículo de una marca en concreto. Se incluyen en el coche cuando es comprado y no se pueden modificar una vez instaladas. Esto es, no existen apenas en el mercado soluciones portables, universales y válidas para su instalación en cualquier dispositivo móvil terrestre.

De todas las tecnologías ADAS existentes, una de las más susceptibles de convertirse en un sistema portable es la detección de agentes de la vía mediante visión por computador. Esto es debido sobre todo al desarrollo de la tecnología para dispositivos móviles, ya que cámaras de pequeño tamaño y alta resolución existen en el mercado desde hace ya bastante tiempo.

En el ámbito de los smartphones no existen apenas soluciones basadas en visión por computador ya que, hasta hace muy poco tiempo, estos dispositivos aún no tenían la capacidad de procesamiento suficiente como para mover este tipo de aplicaciones. Hoy en día esto ha cambiado, y para aprovecharlo, se ha desarrollado el presente proyecto.

1.2. Descripción del proyecto y objetivos

Este proyecto consiste en el desarrollo de una aplicación para el entorno Android capaz de detectar y realizar un seguimiento de peatones a través de la cámara de un smartphone. Además se ha incluido una biblioteca anteriormente creada que posibilita la estimación de la distancia a dichos peatones. En las etapas finales del proyecto se han extendido estas mismas funcionalidades aplicadas en la detección de coches.

Para la detección de los peatones y coches se han utilizado los detectores Haar Cascade y HOG presentes en las librerías de OpenCV. Para realizar el seguimiento de los objetos detectados se utiliza un filtro de Kalman junto con el “Hungarian Method”, utilizado para obtener la correspondencia entre nuevos objetos detectados y antiguos tracks anteriormente en seguimiento. Los detalles de este proceso pueden consultarse en apartados posteriores. Como ya se ha indicado, la biblioteca de funciones que permite estimar la distancia a los objetos detectados ya estaba diseñada. Sin embargo, al incluirla en el presente proyecto, se han optimizado algunas funciones y se han incluido otras para poder adaptarla mejor a este proyecto.

Hasta hace unos pocos años el desarrollo de una aplicación como la que presenta este proyecto era prácticamente impensable. Para empezar no existían bibliotecas de visión por computador disponibles para trabajar sobre dispositivos móviles. Además, estos dispositivos no contaban con la capacidad suficiente como para poder manejar de manera correcta aplicaciones de este tipo. No obstante, los continuos avances en el desarrollo de la electrónica han llevado a la creación de procesadores para smartphones de hasta 8 núcleos y 2 GHz de velocidad por núcleo. Asimismo es fácil encontrar en el mercado teléfonos móviles dotados con cámaras de alta resolución y que graban a cámara lenta. Hemos llegado por tanto a un punto en el que la tecnología está sobradamente capacitada para realizar las tareas que este proyecto necesita.

La principal ventaja de la aplicación aquí desarrollada es que se trata de un sistema ADAS totalmente portable y asequible. Es portable porque ha sido diseñada para que funcione colocando el dispositivo móvil en la luna delantera del coche, tal y como se haría con otros dispositivos como el GPS. Esto implica que puede ser utilizada en coches de todo tipo y tamaño, así como en otro tipo de vehículos, como pueden ser los camiones. Además, para que la aplicación funcione, solo es necesario colocar el smartphone en un lugar donde la cámara tenga visión del exterior, por lo que se puede utilizar también en otro tipo de vehículos que no incorporen una luna o cristal delantero.

Se ha mencionado que la aplicación es asequible porque debería ser capaz de ejecutarse de manera correcta en cualquier smartphone de gama media o alta fabricado en los últimos años. Hay que tener en cuenta además que la sociedad de hoy está fuertemente ligada a las tecnologías de la información y al mundo de la informática en general, por lo que lo más común es que la mayoría de personas posea y utilice a diario un smartphone. Esto último implica que el uso de la aplicación no requiere de inversiones adicionales en ningún tipo de material.

Esta aplicación ha sido incluida como parte de otra mayor propiedad del Laboratorio de Sistemas Inteligentes, o LSI por sus siglas, perteneciente al Departamento de Sistemas Inteligentes y Automática de la Universidad Carlos III de Madrid. Esta aplicación base incluye todos aquellos proyectos que se han ido realizando a lo largo del tiempo en el laboratorio en el ámbito de los sistemas ADAS.

Figura 3: Pantalla Principal de la App LSI

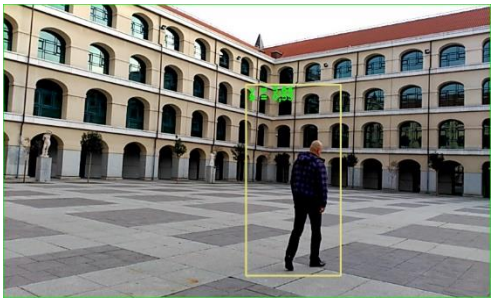


Figura 4: Aplicación de detección de peatones

Este proyecto deja una puerta abierta a otros proyectos que continúen añadiendo más funcionalidades, como pueden ser por ejemplo la de generar alarmas acústicas o de otro tipo cuando se prevea que existe una situación de peligro, o incluir más agentes en el proceso de detección y seguimiento, como ciclistas o motociclistas.

2. ESTADO DEL ARTE

En este apartado se hará un repaso de los sistemas ADAS, tanto en forma de app como de sistemas embarcados en los automóviles, tanto comerciales como académicos, disponibles a día de hoy.

Existen muy pocas soluciones basadas en apps para smartphones en el campo de los ADAS, y muy pocas de ellas utilizan la cámara y la visión por computador. Las que lo hacen se basan la mayoría en la detección de señales de límite de velocidad, y unas pocas de ellas son capaces además de detectar los coches que se encuentran delante de nuestro vehículo. Asimismo apenas existen soluciones basadas en la librería de OpenCV.

Sin embargo, existen multitud de sistemas embarcados cuya función es la de hacer más fácil la tarea de conducción, más productiva, y ante todo, más segura para el conductor y para los peatones. Algunas de ellas, como los sistemas de ayuda al aparcamiento y los sistemas de frenado en caso de emergencia, están basados, al igual que la aplicación desarrollada, en visión por computador. Algunas además incluyen el uso o se basan en visión infrarroja.

2.1. Aplicaciones móviles

A continuación se analizarán algunas aplicaciones disponibles en Google Play relacionadas con los sistemas avanzados de asistencia a la conducción, haciendo especial hincapié en aquellas que están basadas en el uso de la cámara.

2.1.1. iOnRoad Augmented Driving

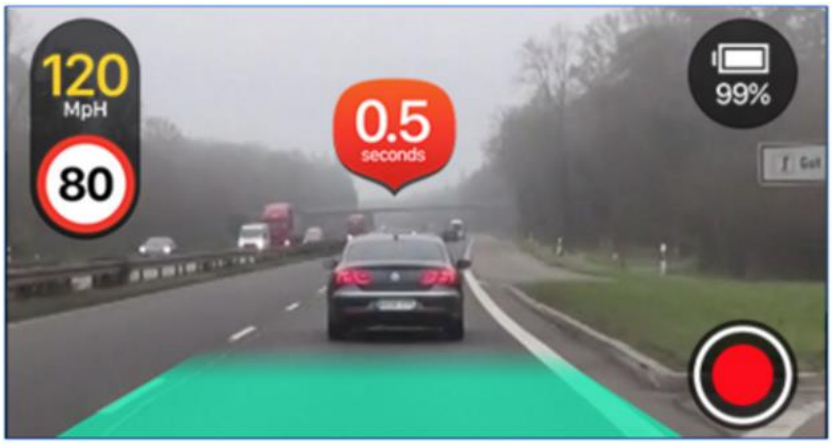


Figura 5: Aplicación iOnRoad Augmented Driving [48]

La función principal de esta aplicación es la de avisar de posibles colisiones. Utiliza la cámara y el GPS del smartphone para detectar los vehículos que tiene delante, así como la distancia a estos y la velocidad a la que nos movemos. Cuando la app determina que existe un peligro inminente dispara una alarma visual para que el conductor pueda responder a tiempo.

Entre las principales características de la aplicación se pueden destacar:

- Advertencia de distancia de seguridad: utiliza la cámara del smartphone para controlar la distancia a la que se encuentran los vehículos que tenemos delante.
- Advertencia de salida de carril: emite advertencias audiovisuales cuando detecta que nos estamos saliendo del carril.
- Localizador de vehículo: detecta automáticamente donde se ha aparcado y hace una fotografía de la plaza de aparcamiento. Más tarde utiliza el GPS para guiar al conductor hasta encontrar dicha plaza.

2.1.2. Road Sign Recognition

Figura 6: Aplicación Road Sign Recognition [49]

El autor de esta aplicación explica que utiliza la cámara del dispositivo para detectar señales de tráfico de límite de velocidad y las destaca en la parte izquierda de la pantalla. Además anuncia el límite de velocidad impuesto por la señal en forma de voz en el idioma que el usuario elija. También se indica que la capacidad de detección y reacción se ve seriamente limitada en condiciones de poca luz, en dispositivos con un procesador lento y si la luna delantera del vehículo tiene gran cantidad de suciedad. Solo detecta señales circulares de contorno rojo.

2.1.3. myDriveAssist

Figura 7: Aplicación myDriveAssist [50]

El creador de esta aplicación es la empresa alemana Bosch, y al igual que la anterior, su función principal es la de detectar y avisar de las señales de limitación de velocidad. Esta además es capaz de detectar las señales de inicio y fin de prohibición de adelantamiento, avisando al conductor cuando se incumplen. Además se conecta a internet para subir a la nube todos los límites que va encontrando, lo que enriquece la experiencia de los demás usuarios.

En la descripción de la app los desarrolladores avisan de que la fiabilidad de la detección varía mucho en función del dispositivo móvil que se utilice y de que la detección se ve muy limitada en escenas nocturnas y a contraluz.

2.1.4. Mobileye 5

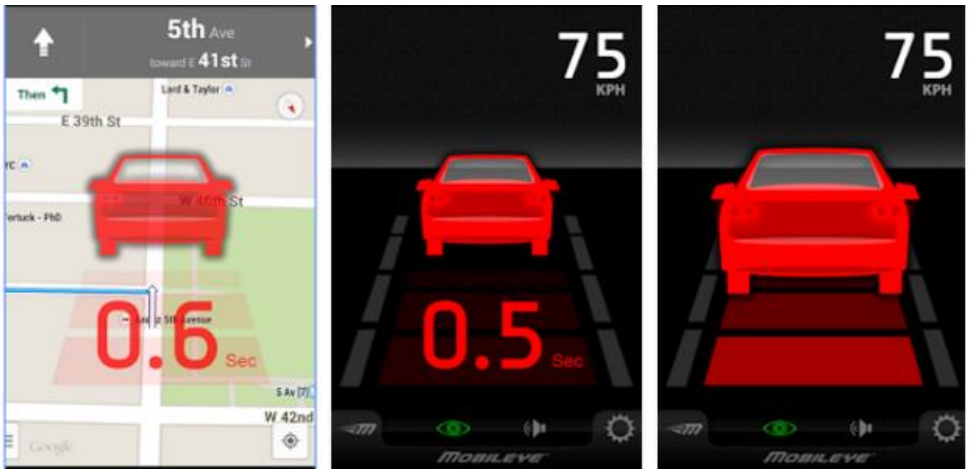


Figura 8: Aplicación Mobileye 5 [51]

Esta aplicación ha sido especialmente diseñada para funcionar únicamente en coches que incorporen hardware de la serie Mobileye 5 [3]. Mobileye es una empresa dedicada a la comercialización de sistemas destinados a evitar colisiones. Para ello utiliza hardware especializado que se comunica con la aplicación para móviles para así poder detectar y analizar coches y peatones, ciclistas, bicicletas, marcas de la carretera y señales de tráfico.

La tecnología empleada en su hardware está basada por completo en el uso de visión por computador con una sola cámara. El procesamiento de imágenes se realiza mediante chips integrados que incluyen unidades de procesamiento especializadas, unidades CPU de uso general para controlar el sistema completo, y entradas y salidas de video y Can bus. En la imagen de la izquierda se puede observar un esquema del hardware utilizado.

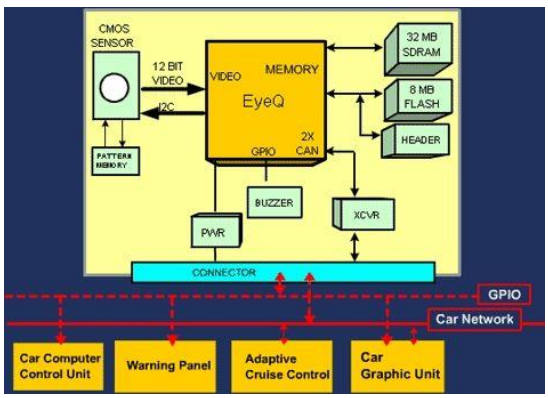


Figura 9: Esquema del hardware Mobileye 5 [3]

La cámara utilizada por el sistema es una cámara digital CMOS con una resolución de 752 x 480 píxeles que trabaja a 60 frames por segundo. Consume 3.3 V trabajando a la máxima resolución y fps.

Para la detección de peatones el sistema utiliza un reconocimiento avanzado de patrones, clasificadores y flujo óptico. Para discernir el movimiento del fondo del movimiento real de algún objeto, el conjunto se basa en que el flujo de ese movimiento de fondo siempre se expande y se dirige hacia los bordes de la cámara. Cualquier otro tipo de flujo es susceptible de ser un objeto en movimiento. La tecnología es capaz de detectar peatones estáticos y en movimiento a distancias de 30 metros utilizando imágenes de resolución VGA. Se utiliza un clasificador de cuerpo entero, ya que se comprobó que la clasificación por partes fracasaba a la hora de detectar peatones a grandes distancias. El sistema solo funciona de día, aunque se están desarrollando métodos para extender la detección de peatones al amanecer y al anochecer.

Los coches Volvo S60 y V60 utilizan esta tecnología. El sistema avisa al conductor cuando detecta que un peatón se encuentra en la ruta de circulación y existe riesgo de una posible colisión. Si este no reacciona o la colisión es inminente, se utiliza un sistema de fusión sensorial entre la cámara y un radar para tomar una decisión de frenado.

El conjunto de hardware y la aplicación para móviles es capaz de aportar las siguientes ayudas a la conducción:

- Alerta por colisión con vehículos que se encuentran delante.
- Alerta por colisión con peatones.
- Aviso por incumplimiento de la distancia de seguridad con vehículos delanteros.
- Aviso por salida involuntaria de carril.
- Reconocimiento de señales y aviso por incumplimiento de la velocidad máxima permitida.

2.1.5. aCoDriver 4

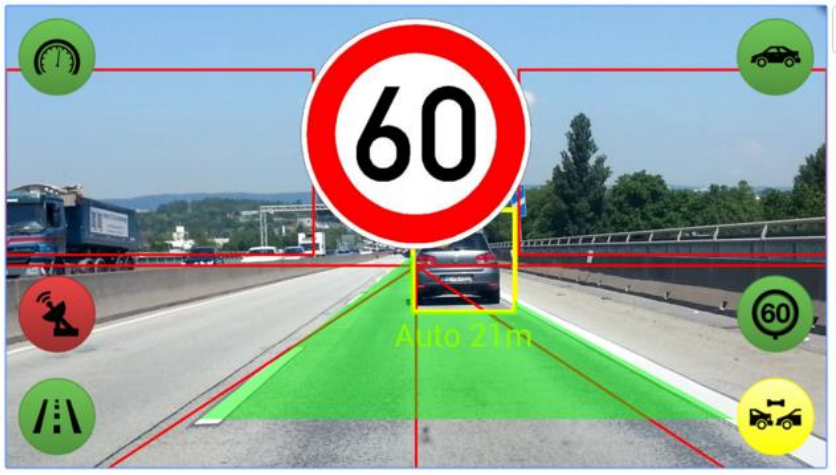


Figura 10: Aplicación aCoDriver4 [52]

La aplicación se basa solamente en el uso de la cámara del móvil y del GPS. Está pensada para funcionar fijando el dispositivo en horizontal a la luna delantera del coche. Las funcionalidades que incluye la aplicación son:

- Detecta y supervisa los límites de velocidad.

- Calcula la distancia entre nuestro vehículo y el delantero, avisando si se sobrepasa el límite de seguridad.
- La aplicación avisa al conductor cuando detecta que el vehículo sobrepasa una línea continua.

Las limitaciones de la aplicación son las mismas que para otras similares. Esto es, la eficiencia en la detección cae en escenas nocturnas, cuando el smartphone posee un procesador lento o cuando la cámara trasera tiene una resolución inferior a VGA.

En cuanto a aplicaciones que utilicen únicamente el móvil para detectar peatones no se ha encontrado ninguna. Las únicas apps disponibles en Google Play que tratan el tema de los peatones se limitan a encender la pantalla con colores vivos cuando detectan que estos están cruzando un paso de cebra, o son apps que calculan la velocidad a la que caminan otros peatones indicando al teléfono cuando comienzan a caminar y cuando paran. Ninguna detecta peatones.

2.2. ADAS

Los objetivos principales de estos sistemas son mejorar la seguridad, la eficiencia y el confort del transporte. Para ello estos sistemas mejoran la funcionalidad de los vehículos e infraestructuras haciendo uso de las tecnologías de la información, trabajando a nivel de vehículo, infraestructura, comunicación entre vehículos y comunicación entre vehículos e infraestructura.

Los ADAS se pueden dividir en dos grupos: sistemas pasivos y sistemas activos. Los sistemas pasivos son aquellos que, una vez se ha producido el accidente, intentan mitigar en la medida de lo posible los daños causados tanto a los ocupantes del vehículo, como a otros agentes de la carretera involucrados, como pueden ser los peatones. Por su parte los sistemas activos están diseñados para evitar que se produzca el accidente. Como ejemplos de sistemas pasivos encontramos el cinturón de seguridad, el Airbag, los reposacabezas activos o los sistemas de capó activo. Entre los sistemas activos contamos con tecnologías como el control de crucero inteligente, la alerta de cambio involuntario de carril o el asistente de ángulos muertos entre muchos otros.

Este tipo de tecnologías suelen requerir grandes inversiones de capital y tiempo por parte de las empresas comercializadoras de vehículos, ya que su función principal es la de mejorar la seguridad en la conducción. Cualquier error de detección puede llevar a que el sistema tome decisiones equivocadas, las cuales pueden provocar consecuencias aún peores que las que produciría un posible accidente por sí mismo.

Los sistemas activos son especialmente útiles para evitar errores de tipo humano, los cuales son con diferencia los más frecuentes. En el 95 % de los accidentes el conductor tiene relación con el suceso, y en el 76 % de las ocasiones es el conductor el único responsable. Las distracciones al volante, que también son una de las principales causas de accidentes, ocupan el 38 % de los casos.

Para poder realizar sus funciones los sistemas ADAS están basados en el uso de los siguientes sensores:

- Radar de largo alcance: utilizado ante todo en el control de crucero inteligente.
- Cámaras de infrarrojos: usadas para la detección de peatones en escenas nocturnas.
- Cámaras de vídeo: se encargan de la detección y clasificación de la información vial (señales, carriles, otras circulaciones, etc.) y de los peatones. Además son el componente principal del asistente de ángulos muertos y de algunos sistemas de aparcamiento.
- Radar de corto alcance: utilizado para la detección de peatones y vehículos cercanos, así como en los sistemas de ayuda al aparcamiento.
- Ultrasonidos: usados sobre todo en sistemas de ayuda al aparcamiento.

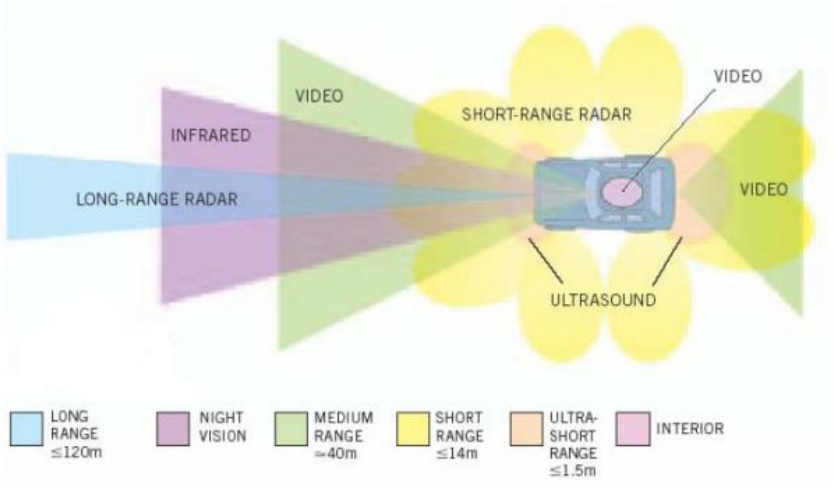


Figura 11: Sensores utilizados en ADAS [60]

A continuación se hará un repaso de algunos de los sistemas avanzados de asistencia a la conducción disponibles en el mercado, montados en automóviles de distintas marcas. Al igual que en el caso anterior, se hará especial hincapié en aquellas tecnologías que utilicen sistemas basados en cámaras, ya sean monoculares, stereo o infrarrojas.

2.2.1. Asistente de ángulos muertos

Figura 12: Asistente de ángulos muertos [53]

Por ángulo muerto se entiende cualquier zona que el conductor no es capaz de ver con la ayuda de los espejos retrovisores interior y exteriores. Este tipo de sistemas suelen contar con cámaras o sensores radar colocados en el espejo retrovisor que vigilan constantemente los ángulos muertos. Si el conductor intenta cambiar de carril y hay un vehículo en el ángulo muerto, el sistema avisa mediante una luz o un sonido de tal situación.

En concreto el asistente de ángulo muerto de Mercedes [4], basado en radar de corto alcance, muestra un mensaje de aviso en el retrovisor exterior en forma de triángulo rojo brillante cuando detecta vehículos en el ángulo muerto. Si además el conductor da el intermitente, se emite una advertencia acústica. El sistema está diseñado para funcionar a velocidades de entre 30 y 250 km/h. Su funcionalidad está además limitada a una velocidad relativa máxima de 16 km/h entre el vehículo detectado y el nuestro.

Continuando con Mercedes-Benz, esta empresa ha diseñado un sistema llamado control activo de ángulo muerto [5] que va un paso más allá, permitiendo al coche actuar en caso de peligro. El sistema muestra en el espejo retrovisor exterior los vehículos situados en el ángulo muerto, y si el conductor se dispone a cambiar de carril, el sistema genera de forma automática una reacción de guiñada para devolver al coche a su carril. Esta reacción consiste en una intervención selectiva en los frenos de las ruedas.

Otro asistente de ángulos muertos montado por Audi en algunos de sus modelos es el llamado Audi Side Assist [6]. En este caso el parachoques trasero del automóvil está equipado con sensores radar que permiten vigilar la presencia de otros vehículos en el ángulo muerto. El sistema se activa de forma automática para velocidades superiores a 30 km/h. Cuando hay un coche en el ángulo muerto o un vehículo se aproxima a gran velocidad por detrás, se activa un LED de aviso en su respectivo espejo retrovisor. Si el conductor da el intermitente en esta situación, la misma luz comienza a parpadear para avisar del peligro. En ningún caso este sistema actúa sobre el control del vehículo.

2.2.2. Sistema de ayuda al aparcamiento

Figura 13: Sistema de ayuda al aparcamiento mediante cámara de video [54]

Esta tecnología hace uso de la información procedente del video trasero, los ultrasonidos y los radares de bajo alcance para realizar su cometido. Existen distintos tipos de sistemas de ayuda al aparcamiento, en función de los sensores utilizados y de la funcionalidad del mismo:

- Hay sistemas que constan únicamente de una cámara trasera, en cuyo caso se utiliza para que el conductor tenga visión de la parte trasera del automóvil desde dentro del vehículo. En algunos casos se utiliza esta cámara además para detectar toda clase de obstáculos, como pueden ser peatones u otros vehículos, y avisar al conductor en caso de exceso de proximidad.
- Los sistemas basados en ultrasonidos normalmente se utilizan para indicar mediante alarmas sonoras la distancia a la que se encuentra el obstáculo trasero más cercano. En algunos modelos también se muestra en una pantalla dicha distancia. Rodeando al vehículo de sensores de ultrasonidos se puede llegar a conseguir que aparque de forma autónoma.
- Por último existen sistemas que se apoyan en el uso de radares de bajo alcance. En este caso es común que el vehículo ea capaz de buscar una plaza de aparcamiento y aparcar en ella por sí solo si el conductor así lo desea.

Por motivos de seguridad, en caso de que el coche sea capaz de aparcar por sí solo, el sistema solo controla la dirección, siendo el conductor el que controla con el acelerador y el freno la velocidad a la que se realiza la operación.

Dentro de este tipo de ADAS se puede destacar el sistema Active Parking Assist [7], usado por Mercedes-Benz en algunos de sus modelos. Cuando el coche circula a una velocidad inferior a 36 km/h, el sistema se activa de forma automática en búsqueda de una plaza de aparcamiento, tanto paralela como perpendicular al sentido de circulación. Cuando una plaza ha sido encontrada, se avisa al conductor mediante una letra P en pantalla. En ese momento, para activar el proceso, el conductor solo tiene que activar la macha atrás y presionar un botón. En este instante se activa el sistema PARKTRONIC, consistente en un total de 10 sensores de ultrasonidos (6 en el parachoques delantero y 4 en el trasero), los cuales inspeccionan la parte trasera y delantera del vehículo. Con esta información el coche calcula el ángulo más favorable de aproximación, actuando sobre la dirección en consecuencia. Como ya se ha indicado, el conductor es quien controla en todo momento la velocidad mediante el acelerador y el freno. Si el vehículo ha sido aparcado de forma autónoma en una plaza, también es capaz de abandonarla de forma automática.

Por su parte Valeo comercializa un sistema de ayuda al aparcamiento universal, disponible para su instalación en cualquier vehículo, llamado beep&park/vision [8]. El conjunto consta de 4 sensores de ultrasonidos colocados en el parachoques trasero, una cámara de video en color para instalar en este mismo parachoques, el cableado necesario, una pantalla LCD en color para instalar en el salpicadero, la centralita de control y todos los accesorios necesarios para su montaje y funcionamiento. Los sensores están especialmente diseñados para poder ser instalados en cualquier vehículo sin afectar a la estética. Cuando el conductor engrana la marcha atrás, automáticamente se activan los sensores de ultrasonidos y la cámara. Los ultrasonidos se utilizan para indicar mediante una alarma sonora la distancia a la que se encuentran los obstáculos, a la vez que la pantalla LCD visualiza las imágenes captadas por la cámara.

Opcionalmente se pueden instalar otros 4 sensores de ultrasonidos en el parachoques delantero.

2.2.3. Alerta por cambio involuntario de carril

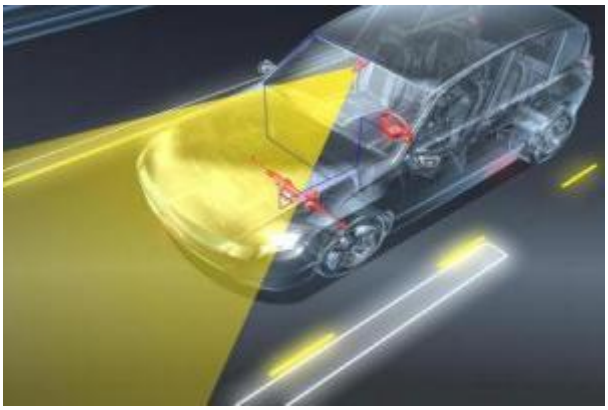


Figura 14: LDW (Line Departure System) [55]

Este tipo de sistemas avisan al conductor mediante algún tipo de alarma de un cambio involuntario de carril. Como norma general, suelen estar basados en visión por computador y únicamente entran en funcionamiento cuando el coche supera los 60 – 80 km/h. El hecho de estar basados en visión por computador lleva asociado dos desventajas. La primera consiste en que la capacidad de cómputo necesaria para analizar el trazado es muy grande, lo que ocasiona que estos sistemas tengan un precio elevado. El segundo inconveniente es que su efectividad se ve seriamente reducida en condiciones de baja visibilidad.

El sistema de alarma suele ser una vibración en el volante o en el asiento, así como señales acústicas y luminosas. Este tipo de alarmas, sobre todo las basadas en vibración, tienen el inconveniente de que pueden provocar movimientos muy reactivos. Esto, unido a que en la mayoría de las ocasiones la salida del carril se produce por un estado de somnolencia, puede provocar movimientos y acciones bruscas sobre los pedales o la dirección por parte del conductor, agravando las consecuencias de un posible accidente.

Existe otro tipo de sistemas que van un paso más allá, llamados sistemas de ayuda de mantenimiento de carril. En este caso el sistema es capaz de actuar sobre la dirección para devolver al vehículo a un rumbo seguro, impidiendo que abandone el carril.

Toyota posee un sistema de ayuda de mantenimiento de carril llamado Lane Keeping Assist [9]. Detrás del espejo retrovisor interior se encuentra una cámara capaz de reconocer algunos tipos de líneas de carretera (en concreto líneas blancas y amarillas). Con esta información es capaz de reconocer la estructura de la carretera y puede actuar sobre la dirección en caso de necesidad. El sistema consta de dos componentes. El primero es un sistema de alarma que avisa al conductor mediante un zumbador, una luz de aviso y una fuerza sobre el volante que intenta devolver el vehículo a su carril. El segundo componente entra en juego cuando el conductor no toma medidas ante las señales de alarma, y consiste en la activación del control de crucero. Desde este momento se aplica una pequeña fuerza en la dirección, contraria a la actual, que

intenta mantener el vehículo centrado en el carril. El principal inconveniente de este sistema, aparte de los anteriormente mencionados, es que no entra en funcionamiento en carreteras no marcadas o cuyas marcas estén deterioradas.

La mayoría de fabricantes que comercializan este tipo de tecnología se basan en el uso de una cámara situada en el espejo retrovisor interior, pero este no es el caso de Citroën [10]. Desde 2005 ofrece en los modelos C4, C5 y C6 un sistema de aviso por salida de carril basado en un sensor infrarrojo situado sobre las ruedas delanteras. Con esto consigue detectar las transiciones blanco-negro que se producen al pasar sobre las líneas que separan los carriles. El sistema de alarma consiste en este caso en una vibración del asiento. Al igual que en el caso anterior, el sistema no funciona en carreteras sin marcar.

2.2.4. Detección de señales de tráfico

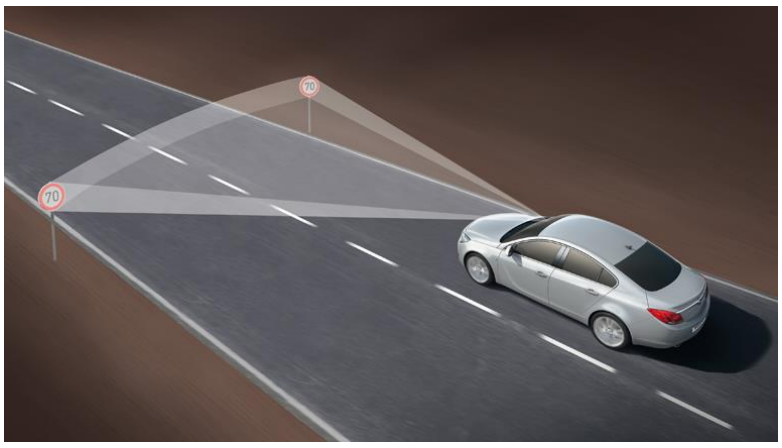


Figura 15: Sistema de detección de señales de tráfico [11]

El funcionamiento de esta tecnología está basado en el uso de una cámara de video colocada en la parte superior de la luna delantera del coche, detrás del espejo retrovisor interior. Con ella el vehículo es capaz de detectar y clasificar las señales de tráfico. Tras la detección y reconocimiento de una determinada señal, el sistema la muestra en alguna pantalla, situada normalmente en el salpicadero o detrás del volante. Si la señal es de límite de velocidad, se emite una alarma en caso de sobrepasar dicha velocidad. Para una correcta detección de las señales es necesario utilizar cámaras con una resolución relativamente alta, por lo que la capacidad de cómputo necesaria es también alta y su instalación en un vehículo puede llegar a suponer un valor extra de hasta 500 €.

Es muy complicado diseñar un sistema global de detección de señales debido a que cada país cuenta con una señalización diferente. Por este motivo, lo más común es que los sistemas comerciales disponibles a día de hoy solo estén diseñados para detectar las señales de limitación de velocidad a partir de 60 km/h, así como las señales de inicio y fin de prohibido adelantar. Además la cámara instalada suele aprovecharse para realizar otras operaciones, como pueden ser el activar las luces cuando es de noche, activar el limpiaparabrisas si llueve o formar parte del sistema de alerta de cambio involuntario de carril.

En cuanto a sistemas comerciales tenemos a Opel, empresa que ha equipado a los modelos Insignia, Corsa y Astra con el llamado Opel Eye [11]. Una cámara situada tras el espejo retrovisor interior se activa para velocidades comprendidas entre 14 y 200 km/h. Detecta las señales de límite de velocidad y de inicio y fin de prohibido adelantar, y las muestra en el display central, tras el volante. La cámara además se utiliza para el sistema de alerta de cambio involuntario de carril, el indicador de distancia de seguridad, el sistema de alerta de colisión frontal y el asistente de luz de carretera.

Algunos modelos de BMW, como el BMW 740d, también incluyen un sistema similar al ofrecido por Opel. BMW y Mercedes fueron los primeros en comercializar vehículos dotados con un sistema de detección de señales (en 2008 y 2009 respectivamente), siendo BMW el primero que lo hizo incorporando datos de mapas y teniendo en cuenta las condiciones de circulación (tráfico, nocturnidad, etc) y las condiciones climatológicas. En un estudio realizado por RACC en colaboración con ADAC [12] se comprobó que el sistema de BMW es superior en cuanto a detección de señales, en comparación con todos sus competidores. El sistema alcanzó un 92 % de tasa de reconocimiento. Además BMW es el único fabricante que incluye un sistema de proyección de la información captada en el campo visual del parabrisas delantero del coche.

El mayor reto en el campo de la detección de señales de tráfico está actualmente en diseñar un sistema que sea capaz de extraer información de los paneles informativos, ya que estos contienen texto y una enorme variedad de símbolos distintos. Los sistemas en desarrollo a día de hoy se basan en los siguientes conceptos:

- Modelos deformables y algoritmos genéticos para la detección de los paneles.
- Sistemas OCR (optical character recognition) para la extracción del texto.
- Redes neuronales para la detección y clasificación de símbolos.

2.2.5. Sistema de detección de peatones

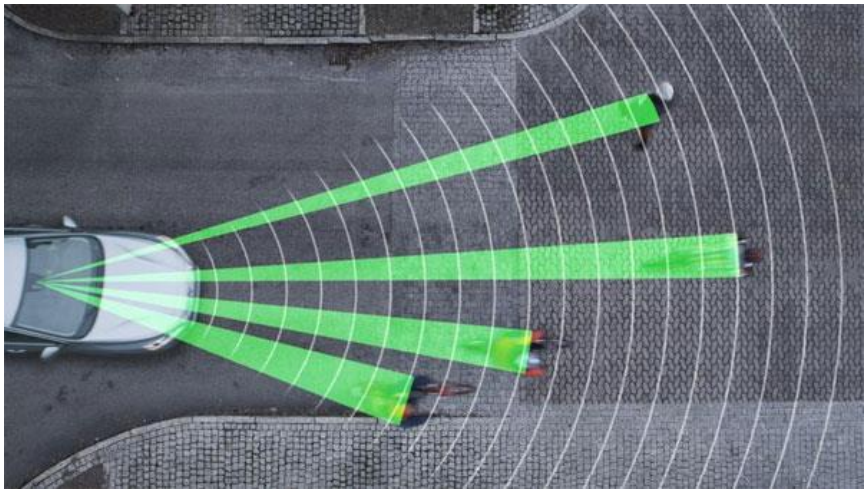


Figura 16: Sistema de detección de peatones y ciclistas [56]

Según se apuntaba anteriormente en la introducción de esta memoria, los peatones junto con los motociclistas constituyen los elementos más vulnerables de la circulación. Además las personas contamos con una gran variedad de apariencias y movimientos, lo que dificulta

enormemente la labor de estos dispositivos. Esta tecnología utiliza las imágenes de una cámara situada normalmente tras el espejo retrovisor interior para detectar los peatones y avisar en caso de una posible colisión. Estos sistemas además están capacitados para detener el coche en caso de peligro inminente. La cámara utilizada puede ser monocular o binocular, siendo este último caso el que más se utiliza y el que mejores resultados ofrece. Para detectar correctamente a los peatones el sistema de procesamiento busca simetrías verticales (característica muy representativa de los humanos, pero también de otros elementos de la vía como farolas y señales) y realiza un análisis de contornos. Si la cámara es binocular, se genera un mapa de disparidad, pues un cambio grande (lo que implica un cambio repentino de distancias) suele significar la presencia de un obstáculo.

Este tipo de sistemas cada vez tienen más importancia entre las distintas marcas de coches. Tanto es así que, desde este año, es imprescindible que el automóvil cuente con algún tipo de sistema de parada de emergencia en caso de colisión con peatones para que el test EuroNCap otorgue 5 estrellas a dicho modelo. El test EuroNCAP es un organismo europeo independiente creado en 1997 que promueve y comprueba los sistemas de seguridad de los vehículos comercializados en Europa.

Como ejemplo de uso de esta tecnología tenemos a Volvo con su sistema de detección de peatones y ciclistas [57]. Para realizar dicha detección el sistema cuenta con un radar colocado en el parachoques delantero y una cámara de alta definición colocada en la parte superior de la luna delantera del vehículo. El radar identifica que existe un peligro, mientras que la cámara determina el tipo de peligro detectado. A través de estos dos sensores el coche es capaz de detectar peatones y ciclistas que invadan la trayectoria, avisando al conductor de tal situación, y aplicando un frenado de emergencia si este no toma acciones. Desde Volvo indican que se trata de un sistema de seguridad auxiliar que no alcanza el 100 % de detecciones. Las principales restricciones del sistema consisten en que el peatón debe estar completamente definido, es decir, no puede tener grandes partes de su cuerpo ocluidas, además el sistema no funciona de noche, en túneles y en calles con alumbrado artificial. El peatón debe medir más de 80 cm para su correcta detección.

Otro sistema similar también propiedad de Volvo es el llamado Volvo City Safety [58]. En este caso el vehículo cuenta con un sensor LIDAR láser ubicado junto a la cámara, tras el espejo retrovisor interior. Con él se barre un área de 6 metros delante del vehículo en busca de obstáculos (identificando su posición, su movimiento y la distancia a la que se encuentran), siendo la cámara, al igual que en caso anterior, la responsable de determinar el tipo de obstáculo encontrado por el láser. Su primera versión funcionaba para velocidades de hasta 30 km/h, mientras que la segunda generación (incluida en todos los modelos a partir de 2014) amplia esta velocidad hasta los 50 km/h. Si la colisión es inminente o si el conductor no actúa, automáticamente se acciona el freno y se para el vehículo.

El gran reto de estos sistemas es que, al estar basados en visión por computador, no son capaces de detectar nada en escenas nocturnas o dentro de túneles. Para ello existen otras soluciones basadas en este caso en cámaras de visión infrarroja. Estas soluciones detectan el calor emitido por distintos cuerpos y aplican modelos probabilísticos para discernir cuales de estas fuentes de calor pertenecen a peatones. Al igual que en el caso anterior, la detección es

muy complicada de realizar, en este caso debido a que las personas no son los únicos elementos de la vía que emiten calor (coches, farolas, paneles luminosos y otros también lo hacen).

Honda fue el primer fabricante de turismos que incorporó en su modelo Honda Legend, en 2004, un sistema de visión nocturna llamado Intelligent Night Vision System [59]. La tecnología utiliza cámaras en el infrarrojo lejano para detectar peatones que se encuentren o se dirijan hacia la trayectoria del vehículo. Los vehículos equipados con esta tecnología cuentan con dos cámaras infrarrojas situadas a derecha e izquierda en el parachoques delantero, en la parte baja de este. A partir del tamaño y de la forma se consigue discernir entre peatones y otro tipo de elementos presentes en la vía que generan calor. Si se detecta una situación de peligro, se avisa al conductor mediante alarmas visuales y auditivas. Los modelos que incorporan este sistema de ayuda llevan incorporado en el salpicadero una pantalla que muestra la imagen reconstruida captada por las dos cámaras y un recuadro alrededor de cada peatón detectado. En la imagen inferior se puede observar un gráfico con el flujo de control del sistema utilizado por Honda.

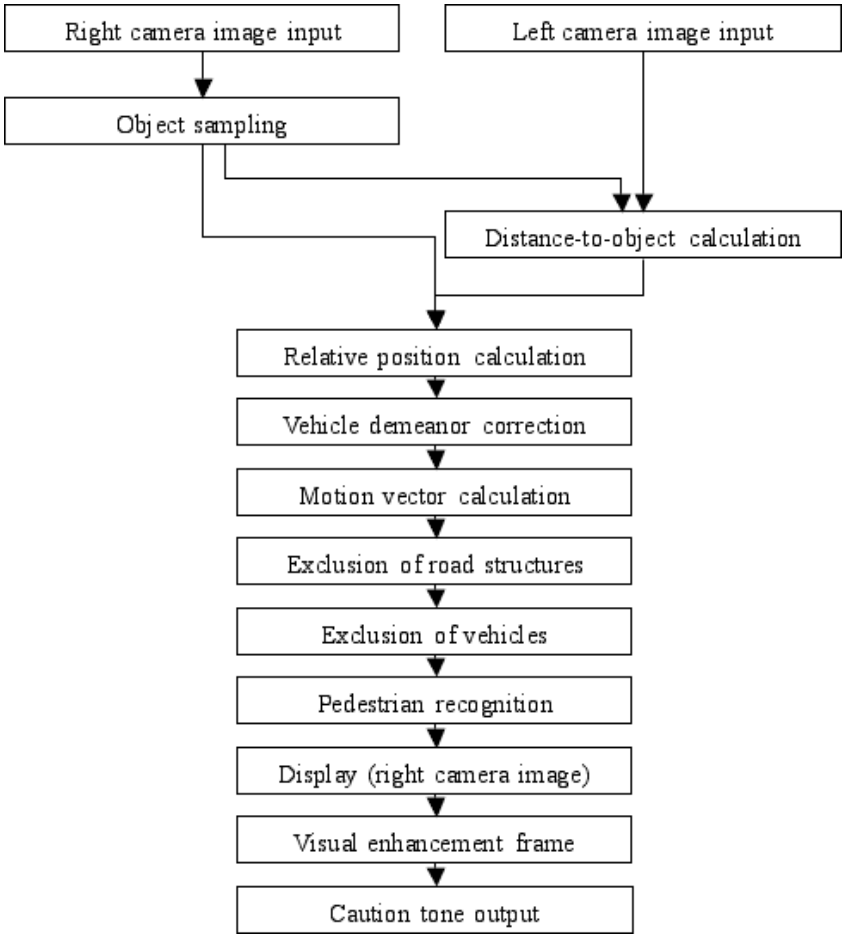


Figura 17: Flujo de control del sistema Honda Intelligent Night Vision [59]

2.2.6. Coche autónomo de Google

No se puede finalizar este repaso de distintas tecnologías ADAS disponibles en el mercado sin mencionar al coche de Google, ya que posiblemente sea uno de los vehículos tecnológicamente más avanzados que existen. Hay que indicar que, evidentemente, el objetivo

de Google no es entrar en el mercado de los turismos, sino que está desarrollando su tecnología para que, cuando esté completamente lista, la pueda vender a las marcas que así lo deseen. Por otra parte, y debido a que el coche dispone de sensores como el Velodine que son extremadamente caros, hasta ahora no ha conseguido casi nada al respecto.

Figura 18: Google Car (2009) [61]

En 2009 Google comenzó a realizar las primeras pruebas con su tecnología de conducción automática [13], montada sobre el vehículo de la figura superior, el Toyota Prius. Las pruebas se realizaron en su mayoría en carreteras de California. Para desarrollar esta tecnología Google contrató al Alemán Sebastian Thrun, de la Universidad de Stanford, y a Chris Urmson, de la Carnagie Melon University. Ambos eran los máximos responsables del departamento de vehículos inteligentes de sus respectivas universidades. Ambas encabezan a nivel mundial el campo de los vehículos inteligentes. Sebastian Thrun, actual jefe del proyecto, diseño junto con su equipo de la universidad el vehículo robótico Standley. Este fue el ganador de la DARPA Grand Challenge en 2005, por lo que recibieron un premio de 2 millones de dólares. El Grand Challenge de 2005 fue un concurso organizado por el ejército de EEUU en el cual los distintos equipos participantes tenían que ser capaces de diseñar un vehículo inteligente que fuese capaz de completar un recorrido de 131,6 millas de forma autónoma. De los 23 finalistas, 5 llegaron a la meta. El Sandstorm, diseñado por el equipo de Chris Urmson, quedó en un segundo puesto.

Los principales sensores y tecnologías incluidos en el Google Car son: varios dispositivos radar, un avisador de cambio de carril, un LIDAR (llamado Velodine, situado en la parte superior), cámaras de visión stereo (para detectar peatones y ciclistas y otros obstáculos), cámaras infrarrojas, GPS con sensor inercial y encoders en cada rueda. Además, claro está, de toda la parte de procesamiento de la información y conexionado. El Velodine consta de un conjunto de láseres en los que cada uno realiza un barrido de 360º a su alrededor. Cada uno de estos sensores genera un plano. Posteriormente toda esta información se dirige a un software en tiempo real que realiza una representación 3D del entorno. En concreto el Velodine del Google Car está equipado con 64 láseres. El sistema en total vendría a costar unos 130.000 € y resulta bastante antiestético, por lo que su implantación en futuros turismos no es viable.

Posteriormente, en 2012, en Google comenzaron a realizar pruebas con una versión modificada del Lexus RX450h, llegando a conducirlo con éxito de forma autónoma en calles de ciudad. Más tarde, en 2014, Google comenzó a desarrollar desde cero un nuevo prototipo de coche totalmente autónomo. En diciembre de 2014 se fabricó dicho prototipo, cuya imagen se puede observar en la imagen inferior.

Figura 19: Google Car (2014) [13]

Pero Google no es la única empresa que avanza rápidamente en el campo de los coches autónomos. En este caso podemos destacar el fabricante de turismos sueco Volvo, con su tecnología Drive-Me [14]. Volvo se ha autoimpuesto el objetivo de que para 2020 sus conductores no sufran muertes ni accidentes graves con ninguno de sus modelos. Para ello está desarrollando la tecnología anteriormente mencionada, la cual es capaz de permitir una circulación completamente autónoma y desatendida por parte del conductor. La empresa asegura que sus coches de conducción autónoma ya circulan por algunas carreteras de Suecia, pero no será hasta 2017 cuando se empiecen a utilizar 100 modelos piloto por vías públicas de todo el mundo.

Por último se puede destacar a la empresa de vehículos eléctricos Tesla Motors, la cual ha desarrollado ya un sistema de conducción semiautomática [69]. Los modelos S y X cuentan con un modo de piloto automático en el que se pueden separar las manos del volante. No obstante, es recomendable mantener al menos los dedos apoyados en el volante por seguridad, además si no se hace el coche avisa. La tecnología presente detrás de este sistema de piloto automático se denomina Autosteer. Los modelos antes mencionados cuentan con un radar frontal, una cámara con capacidad de reconocimiento de imágenes y un sonar de 360º que permiten detectar las líneas de la carretera y otros obstáculos. El modelo S además cuenta con una tecnología denominada Auto Lane Change, la cual permite al vehículo cambiar de carril de forma autónoma con solo activar el intermitente correspondiente. Tesla no es pionera en el campo de los coches autónomos, sin embargo, tiene algo que hace a sus vehículos únicos: cada vez que se realiza un avance en este campo por parte de la compañía, se envía una actualización software a todos los coches de la marca compatibles (modelos a partir de septiembre de 2014), por lo que todos se encuentran actualizados con las últimas características en todo momento.

Uno de los mayores problemas a los que se enfrentan estas tecnologías de conducción autónoma consiste en que la legislación de la mayoría de territorios en el mundo no permite aun el uso de vehículos autónomos, sobre todo en entornos urbanos.

2.3. Tecnologías ADAS en el entorno académico

Las grandes empresas comercializadoras de turismos no son las únicas que trabajan duramente para avanzar hacia el coche autónomo. Las universidades de Stanford y la Carnagie Melon University, anteriormente mencionadas, son claros ejemplos de ello. Es por esto que a continuación se hará un repaso de distintas tecnologías ADAS desarrolladas en el entorno académico.

2.3.1. LSI UC3M: detección nocturna de peatones

Se trata de la tesis doctoral del investigador de la UC3M Daniel Olmeda Reino [15]. En dicha tesis, completada en 2014, el objetico era desarrollar un sistema de detección nocturno de peatones, utilizando para ello cámaras que operan en el infrarrojo lejano. La ventaja de estas cámaras es que no necesitan luz para funcionar, al contrario de lo que ocurre con las cámaras en el infrarrojo cercano o aquellas que utilizan la luz visible.

Figura 20: Coche IVVI de la UC3M equipado con cámaras infrarrojas [16]

El vehículo está equipado con una cámara infrarroja, situada sobre un espejo retrovisor, como se puede apreciar en la imagen superior. El motivo de esta colocación reside en que la luz infrarroja de estas cámaras no es capaz de atravesar los cristales, con lo que deben estar colocadas obligatoriamente fuera del coche. La principal desventaja de esto es que sufren más las inclemencias del tiempo, por lo que tienen que ser más robustas y caras, y se pueden generar problemas de vandalismo fácilmente.

El vehículo es capaz de detectar peatones a una distancia de hasta 40 metros, distancia que podría aumentar si se utilizasen cámaras de mayor longitud focal [16]. Para que el sistema pueda discernir entre lo que es un peatón y cualquier otro elemento de la vía que genere calor, el algoritmo de detección se basa en ciertas características de la silueta invariantes ante cambios en la temperatura y el contraste.

2.3.2. Stanford University: Standley

Figura 21: Vehículo robótico Stanley de la Universidad de Stanford [17]

Como ya se ha mencionado anteriormente en esta memoria, el Stanley es el coche robótico con el cual se presentó y ganó la Grand Challenge (2005) la Universidad de Stanford [17]. El jefe de equipo, Sebastian Thurn, fue contratado por Google y ahora es el jefe del proyecto Google Car. La prueba, organizada por el ejército de los EEUU, pretendía promover la investigación en el campo de los vehículos autónomos. Para poder superar la prueba, los vehículos tenían que circular por una zona de desierto (aunque ciertas zonas estaban asfaltadas) de 175 millas en menos de dos horas, sin ninguna intervención humana.

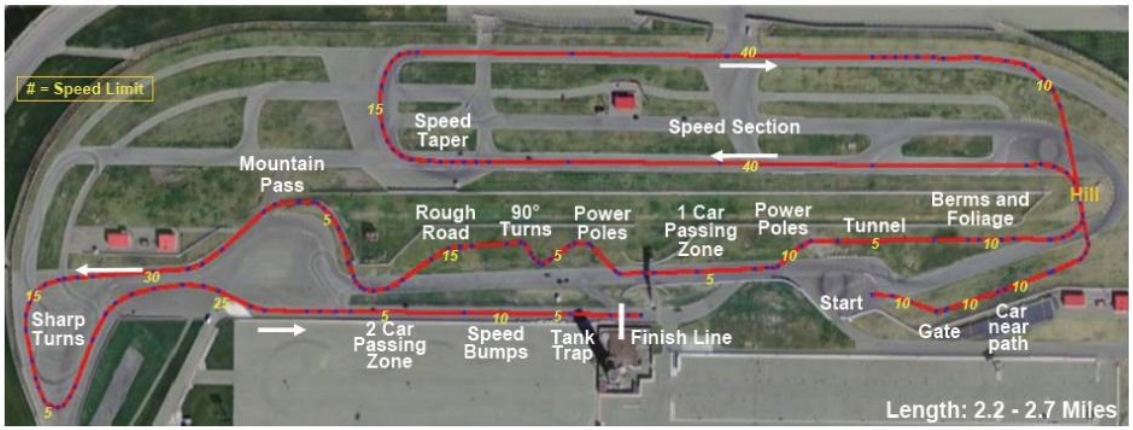


Figura 22: Circuito del Grand Challenge 2005 [62]

El coche está basado en el Volkswagen Touareg R5. El sistema incluye como principales sensores un GPS, una unidad de medida inercial de 6 grados de libertad y un velocímetro en cada rueda. Para el reconocimiento del entorno el vehículo cuenta con cuatro sensores LIDAR en la parte superior (tipo Velodine), un radar, una cámara stereo y una cámara monocular. Toda esta información sensorial se fusiona a una velocidad de 10 Hz, con lo que el coche puede salvar obstáculos fácilmente.

Hoy en día el Stanley se encuentra en el Museo Nacional Smithsonian de Historia Americana. Forma parte de una exposición sobre el futuro de las carreteras y los coches inteligentes.

3. DESCRIPCIÓN GENERAL DEL PROYECTO

3.1. Propósito y características generales del proyecto

El propósito del proyecto es crear una aplicación que sea capaz de detectar y realizar un seguimiento de peatones y coches en el entorno Android. Además este proyecto incluye, haciendo algunas optimizaciones en el código, una librería ya escrita que realiza una estimación de la distancia a la que se encuentra aquello que se ha detectado, utilizando para ello la cámara del smartphone y el acelerómetro (para medir el ángulo de inclinación del dispositivo). Además se ha separado la aplicación en cuatro subaplicaciones. La primera registra la posición de los peatones en coordenadas U y V, es decir, en coordenadas de la imagen, o lo que es lo mismo, el seguimiento se basa en el registro de las coordenadas en píxeles en las que se detecta a la persona. La segunda se basa en esto mismo, pero aplicado en la detección de coches. Por su parte, las subaplicaciones tercera y cuarta registran la posición de peatones y coches basándose en coordenadas X (desviación horizontal respecto al centro de la imagen) e Y (profundidad), ofrecidas por el sistema de estimación de distancias.

Para poder realizar la detección de peatones se podían utilizar, en un principio, tres algoritmos diferentes extraídos de la librería OpenCV. Estos eran: el detector Haar Cascade, HOG y el detector HOGCascade. Esto es así porque OpenCV incluye de forma predeterminada archivos de entrenamiento para la detección de peatones utilizando estos tres detectores. La aplicación se desarrolló inicialmente con la versión 2.4.9 de la librería de OpenCV. Sin embargo, para poder utilizar estas librerías, y por tanto el app, en dispositivos más actuales (equipados con Android 5.0 y superiores), fue necesario actualizar a la versión 3.0.0. RC1. El problema reside en que esta última versión no incluye el detector HOGCascade, por lo que se encuentra desactivado en la aplicación final.

Según indican los desarrolladores de OpenCV, el motivo por el que han eliminado este último detector se debe a que “la implementación de características HOG es poco convencional, diferente de la implementada por Dalal [18] y diferente de la implementada por P. Dollar [19]” [20]. Además el creador de este algoritmo ya no colabora con el equipo de desarrollo de OpenCV. En realidad no es problema demasiado importante, ya que HOGCascade presentaba una tasa de detección buena, comparable o incluso superior en algunos casos a la obtenida con el resto de detectores, pero la cantidad de falsos positivos generados era ampliamente superior.

En cuanto a la detección de coches se refiere, el único detector utilizado en este caso es Haar Cascade, ya que actualmente OpenCV no incluye ningún archivo de entrenamiento en HOG para coches, ni se ha encontrado uno por la web que mejore el funcionamiento de este.

En cuanto al rendimiento y velocidad de la aplicación, hay que indicar que es mayor cuando se realiza la detección de peatones mediante Haar Cascade, disminuye en un 50% cuando se detectan coches a través del mismo detector y disminuye otro 50% al utilizar HOG para la detección de peatones. La diferencia de rendimiento entre la detección de coches y peatones, utilizando el mismo detector, es bastante acusada. Esto se debe al archivo de entrenamiento utilizado y el tamaño de la ventana de búsqueda, pues son las únicas diferencias apreciables entre el código de uno y de otro. Además se pudo apreciar durante el desarrollo del proyecto

que el rendimiento (entendido como la tasa de fps alcanzada) puede variar hasta en un 50 % dependiendo de si se utiliza el archivo de entrenamiento para peatones ofrecido por OpenCV (más rápido), u otros extraídos de diversas páginas web. La tasa de fps alcanzada en función de los detectores y del dispositivo móvil utilizados se concretará en los siguientes apartados.

Para realizar el seguimiento de peatones se utiliza un Filtro de Kalman lineal. OpenCV incluye uno, pero, como ya se ha indicado, este proyecto comenzó a desarrollarse en la versión 2.4.9, y esta, en su versión para Android, no incluía soporte completo para dicho filtro. Por ello el Filtro de Kalman utilizado es JKalman [21], abreviatura de Java Kalman. Esta librería también incluye otra que permite crear y manipular matrices, pues el filtro está basado en cálculo matricial. OpenCV 3.0.0 RC1 si incluye soporte completo para el Filtro de Kalman en su versión para Android. No obstante, como el rendimiento de JKalman es bueno, la librería es liviana y nunca ha dado problemas, se ha decidido no reescribir gran parte de la aplicación y mantener este último. Un análisis más detallado sobre el filtro puede consultarse en los siguientes apartados.

Las mejoras efectuadas en la librería de estimación de distancias, de la que se partía en este proyecto [70], se basan en la optimización del código y en la inclusión de alguna característica nueva. La optimización del código se basa ante todo en la implementación de excepciones y el control de errores, así como en el uso más eficiente de las funciones creadas, pues muchas de ellas repetían una y otra vez el código ya presente en otras. Por otra parte, la aplicación se basa en el cálculo del vector de gravedad para calcular la inclinación del smartphone. En la librería original solo se usa el acelerómetro para realizar el cálculo de dicho vector, por lo que es necesario incluir un filtro paso bajo simple para poder extraer la gravedad del resto de aceleraciones. Sin embargo, en la página de Android Developers [22] se indica que es preferible utilizar el sensor de gravedad. Este sensor está situado dentro de los “sensores por software”, ya que no es un sensor en sí mismo, sino que se combina por software el giróscopo con el acelerómetro para calcular de forma más precisa el vector de gravedad. No todos los dispositivos incluyen un giróscopo, con lo que la aplicación comprueba antes dicha existencia, y si no está disponible un giróscopo, se utiliza el acelerómetro.

3.2. Hardware

Para la elaboración de este proyecto se han utilizado dos dispositivos móviles, uno de gama baja-media (Sony Xpheria J) y otro de gama media-alta (BQ Aquaris E5 4G). El primer dispositivo se utilizó en las primeras etapas de desarrollo de la aplicación. Después se adquirió el segundo modelo y se abandonó el uso del primero debido a sus prestaciones relativamente bajas. En las figuras inferiores podemos observar la apariencia externa de cada uno de ellos.

Figura 23: Sony Xpheria J (smartphone de gama baja-media) [63]

Las especificaciones de este dispositivo más relevantes para este proyecto son:

- Android OS: 4.1.2 (actualizado desde 4.0.4) Jelly Bean
- Espacio de almacenamiento interno: 4 GB
- Cámara trasera de 5 megapíxeles. La delantera tiene resolución VGA.
- Pantalla de 4 pulgadas
- Memoria RAM: 512 MB
- CPU: Qualcomm MSM7227A Snapdragon S1 de un núcleo a 1 GHz
- GPU: Adreno 200
- Sensores: acelerómetro, pero no giróscopo

Las principales "deficiencias" de este dispositivo, en lo que respecta al uso de esta aplicación, están en su procesador relativamente lento y una memoria RAM bastante justa que provoca que todo el sistema se ejecute con relativa lentitud.

Figura 24: BQ Aquaris E5 4G (smartphone de gama media alta) [64]

En lo que respecta al modelo BQ, estas son sus principales características:

- Android OS: 5.0.2 (actualizado desde 4.4.4) Lollipop
- Espacio de almacenamiento interno: 16 GB
- Cámara trasera de 13 megapíxeles (sensor Sony Exmor RS IMX214 BSI) con dual-flash. La cámara delantera tiene una resolución de 5 Mpx.
- Pantalla de 5 pulgadas IPS HD (1280 x 720)
- Memoria RAM: 2 GB
- CPU: Qualcomm® Snapdragon™ 410 Quad Core A53 hasta 1.2 GHz
- GPU: Adreno 306, 400 MHz
- Sensores: dispone tanto de acelerómetro como de giróscopo.

Este dispositivo no presenta grandes dificultades a la hora de mover la aplicación, excepto en el caso de la detección de peatones mediante HOG, pues los fps alcanzados no son muy altos. En el apartado de resultados se muestra una comparativa del rendimiento de la aplicación en ambos dispositivos y para todos los detectores utilizados, con y sin tracks en seguimiento.

3.3. Software utilizado

En este apartado se hará un repaso de todos los elementos software utilizados para la realización del presente proyecto. Estos son: el sistema operativo Android, la librería OpenCV, Eclipse y Android Studio.

3.3.1. Sistema Operativo Android

Android es un sistema operativo Open Source basado en el núcleo Linux. Comenzó siendo un sistema operativo pensado principalmente para su uso en dispositivos móviles táctiles [23]. Esto incluye smartphones y tablets, además de relojes inteligentes, televisores y automóviles. El proyecto fue desarrollado inicialmente por Android Inc. en 2003, empresa que Google financió hasta 2005, momento en el que la compró. El sistema operativo Android fue presentado oficialmente en 2007, pero no fue hasta octubre de 2008 cuando se presentó el primer smartphone Android, el HTC Dream.

Android se ha convertido en el sistema operativo para móviles y smartphones más popular del mundo [24]. Este SO se ejecuta en cientos de millones de dispositivos y en 190 países de todo el mundo. Además no es solo el SO más popular, sino que también es el que más rápido crece, pues cada día se activan más de un millón de dispositivos Android mundialmente y se descargan de Google Play varios miles de millones de apps cada mes.

Una de las principales quejas por parte de los usuarios hacia el SO de Google es la fragmentación que sufre. Esto se debe a que las actualizaciones no llegan directamente de Google a los terminales, sino que es el fabricante del dispositivo móvil el que la recibe y elije si actualiza o no dicho dispositivo. En muchos casos esta actualización nunca se produce, por lo que en el mercado no existe una versión claramente dominante de Android, sino una gran

cantidad de versiones nuevas y antiguas repartidas entre distintos modelos de smartphones. En la siguiente tabla se puede ver la cuota de mercado de cada una de las versiones utilizadas.

Versión	Nombre	Fecha Distribución	API	Cuota (%)
6.0	Marshmallow	29 de septiembre de 2015	23	0,3
5.1	Lollipop	6 de abril de 2015	22	10,1
5.0	Lollipop	3 de noviembre de 2014	21	15,5
4.4	Kit Kat	31 de octubre de 2013	19	37,8
4.3	Jelly Beam	24 de julio de 2013	18	4,1
4.2	Jelly Beam	13 de noviembre de 2012	17	13,9
4.1	Jelly Beam	9 de julio de 2012	16	11,0
4.0	Ice Cream Sandwich	16 de diciembre de 2011	14, 15	3,3
2.3	Gingerbread	9 de febrero de 2011	10	3,8
2.2	Froyo	20 de mayo de 2010	8	0,2

Tabla 1: Cuota de mercado de las distintas versiones de Android (5/11/2015) [23]

Todas las aplicaciones diseñadas para Android están escritas en el lenguaje de programación Java, sin embargo, estas no se ejecutan en una JVM (JAVA Virtual Machine), pues el rendimiento sería insuficiente y el consumo de energía elevado. En su lugar Android ejecuta las aplicaciones en la máquina virtual Dalvik, diseñada específicamente para este entorno. En comparación con JVM, Dalvik pierde la portabilidad que caracteriza a Java para ganar en rendimiento, minimizando el uso de memoria, y disminuyendo el consumo energético. Cada aplicación se ejecuta en una máquina virtual independiente, siendo tarea del sistema operativo la gestión de estos procesos.

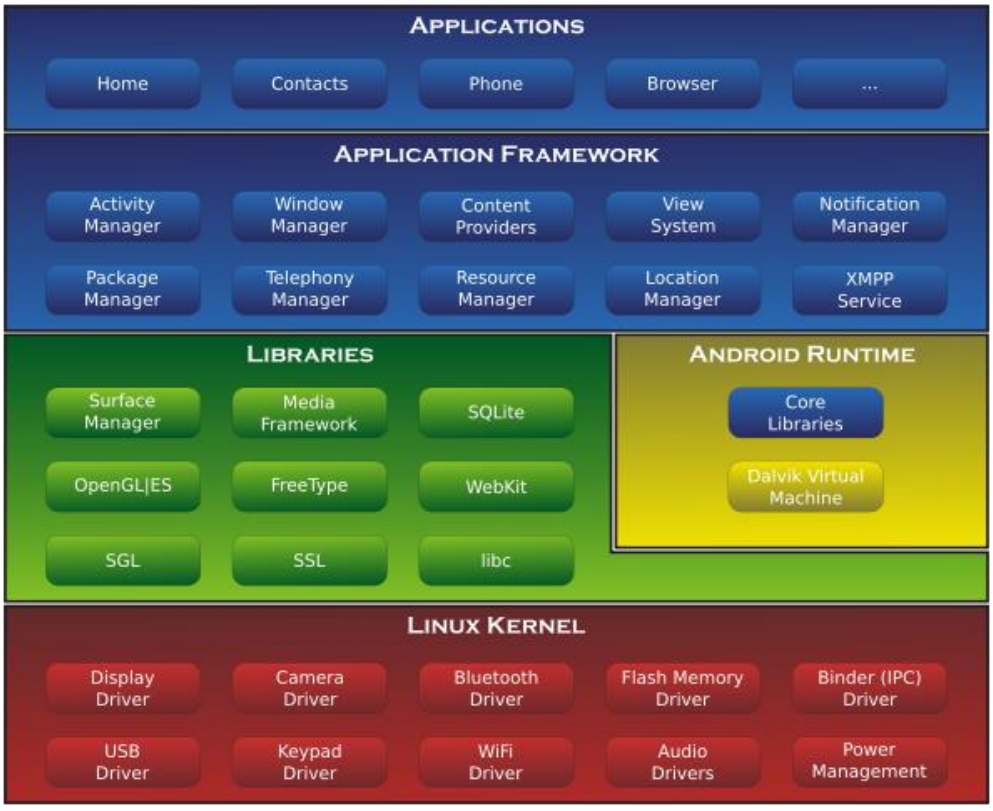


Figura 25: Arquitectura del sistema Android [25]

Como se puede apreciar en la imagen superior (color azul), las aplicaciones no son las únicas escritas en Java, el marco de aplicaciones o framework, que permite acceder a las APIs del sistema operativo, también está escrito en este lenguaje. Android además dispone de una serie de librerías escritas en C/C++ (color verde) como la biblioteca de gráficos y 3D entre otras. Por último, el Kernel Linux tiene las funciones de gestión de seguridad, gestión de memoria y de procesos, pila de red y el modelo de controladores.

Con la llegada de Android 5 al mercado, Dalvik fue sustituido por ART (Android Runtime). Como ya se ha indicado, Dalvik no es una máquina virtual de Java y por tanto no ejecuta bytecode de Java. En lugar de ello el sistema incorpora una herramienta llamada dx que transforma bytecode de Java en archivos .dex ejecutables desde la máquina virtual Dalvik. Sin embargo, el código dex, al igual que Java, es interpretado, lo que significa que las aplicaciones no se compilan cuando se generan, sino que se traducen a un código intermedio entre el original y el binario. Cuando la máquina virtual ejecuta la aplicación, este código se compila y ejecuta al mismo tiempo, reduciendo el rendimiento de la aplicación. La principal ventaja de ART es que el código se compila antes de que la aplicación sea ejecutada, además de mejorar otros aspectos como el colector de basura. Android incluye la herramienta dex2oat para ofrecer total compatibilidad entre el código dex y ELF, utilizado por ART.

Para terminar, la imagen inferior representa el ciclo de vida de una aplicación en Android, algo básico que es necesario comprender cuando se quiere diseñar cualquier aplicación.

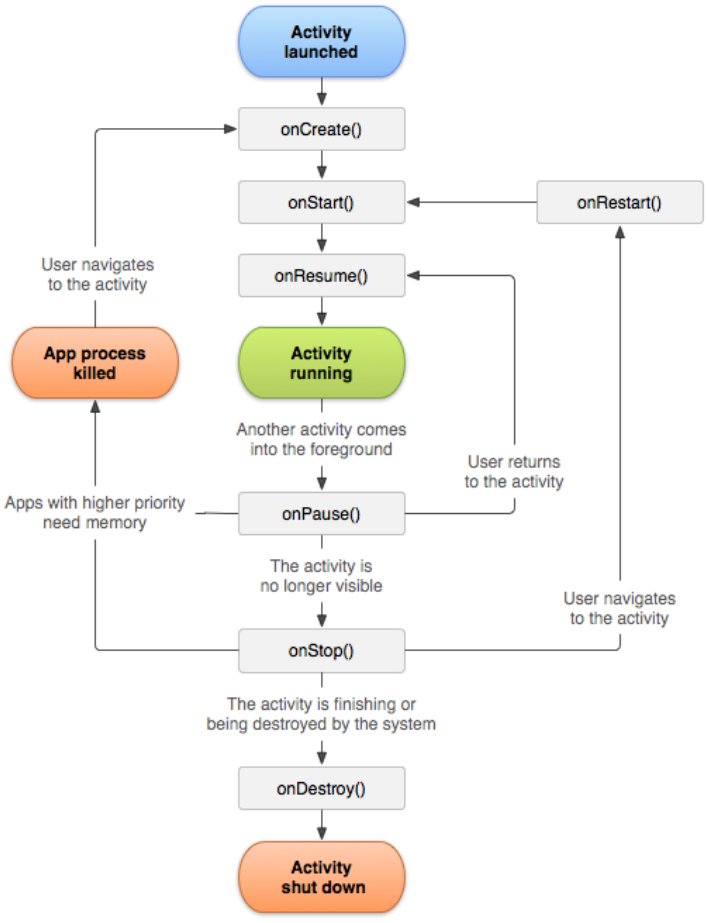


Figura 26: Ciclo de vida de una aplicación Android [27]

La imagen muestra todas las llamadas posibles a distintas funciones que ocurren durante el tiempo que transcurre entre que se inicia la aplicación y se cierra, así como el motivo de esta llamada. Una de las cuestiones más importantes a tener en cuenta es que, desde el momento en que la aplicación deja de estar en primer plano (llamada a onPause()), el sistema operativo es libre de destruir la aplicación si el dispositivo necesita más recursos para otras aplicaciones. Por ello es muy importante guardar los datos más sensibles de la aplicación en la llamada a la función onPause(). Durante su ciclo de vida una aplicación puede estar en tres estados [27]:

- **Resumida**: la aplicación se encuentra en primer plano.
- **Pausada**: hay otra aplicación en primer plano, pero esta es parcialmente visible de alguna forma. En casos extremos de falta de memoria el sistema puede destruir el proceso.
- **Parada**: la aplicación está en segundo plano completamente. Igualmente, si el sistema necesita más memoria, puede ser eliminada en cualquier instante.

3.3.2. OpenCV

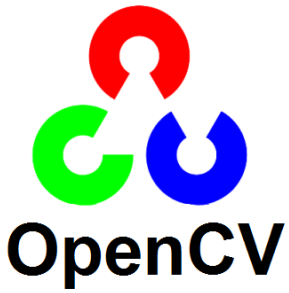


Figura 27: Logo de OpenCV

OpenCV deriva de Open Source Computer Vision. Se trata de una librería de visión por computador y de algoritmos de aprendizaje automático que se distribuye con la licencia BSD, lo que significa que puede ser utilizada de forma gratuita tanto para uso académico como para uso comercial. Comenzó a desarrollarse en 1998 por parte de Intel [28], más tarde el soporte cambió a Willow Garage y actualmente está siendo mantenida por Itseez.

La librería permite trabajar en C, C++, Phyton y Java y además soporta Windows, Linux, Mac OS, Android y iOS [29]. Desde hace algún tiempo los distintos algoritmos utilizados en OpenCV están siendo traducidos a CUDA, lo que permite ejecutar dichos algoritmos en GPUs de Nvidia que posean dicha característica, liberando al procesador de esa carga y aumentando considerablemente el rendimiento de la aplicación. Muchos algoritmos están también diseñados para trabajar en procesadores con varios núcleos, permitiendo su uso en tiempo real. Este es el caso de los detectores Haar Cascade, HOGCascade y HOG, utilizados en este proyecto.

Un detalle a tener en cuenta cuando se utiliza OpenCV en Android es que existen dos formas de acceder a la librería. La primera consiste en instalar una aplicación en el smartphone llamada OpenCV Manager, disponible en Google Play. En este caso este ha sido el método utilizado ya que es la manera de proceder más común, más recomendada por el equipo de OpenCV y más sencilla de implementar. El segundo método consiste en incluir la librería que se desee utilizar en la propia app. OpenCV Manager es una app que inspecciona las características del teléfono y descarga automáticamente las librerías más adecuadas para la arquitectura del procesador, así como las más actualizadas. Existen otras ventajas añadidas como por ejemplo que la aplicación, más concretamente el paquete apk, es más ligero.

La aplicación base, de la que se hablará más adelante, en la cual está incluida la aquí desarrollada, utilizaba la librería de OpenCV 2.4.3. Este proyecto se comenzó a desarrollar en la versión 2.4.9, pues era la recomendada e instalada por OpenCV Manager en el smartphone con

el que se ha trabajado. Sin embargo, tras el lanzamiento de la versión 3.0 gold en junio del año pasado, se actualizó la librería para Android a la versión 3.0. En ese momento se configuró la aplicación para que accediera a la librería actualizada desde OpenCV Manager. La principal ventaja de esta actualización es que la versión 2.4.x no era compatible con Android 5.0 o superior, debido a un pequeño cambio que se produjo en la forma de lanzar procesos en Android con el objetivo de mejorar la seguridad.

Como ya se ha comentado, existen dos formas de utilizar OpenCV con Android [30]. La primera y más común es utilizar la API especialmente diseñada en Java. La segunda consiste en programar en código nativo, es decir, en el código en el que está escrita OpenCV, que es C++. El inconveniente de utilizar el API de Java es que las funciones de OpenCV no están escritas en Java, sino que están compiladas en C++. Para acceder a ellas se utiliza el JNI, el cual permite a los métodos escritos en C++ examinar y utilizar objetos escritos en Java. El problema de esta forma de trabajar es que cada una de estas llamadas a JNI repercute en una pequeña penalización de rendimiento. Existen dos llamadas a JNI por método nativo utilizado, una para acceder al método y otra para convertir a objeto de Java el resultado del método nativo.

El principal inconveniente de programar de forma nativa es que resulta mucho más complicado. Además Java no desaparece por completo, ya que elementos como el GUI siguen estando escritos en este lenguaje. La mejor forma de proceder es encapsulando todo el código de C++ que se quiera utilizar en una misma clase, y llamar desde el JNI a esa clase. De esta forma tenemos solo dos llamadas a JNI, no dos por cada método utilizado. Para proceder de esta forma es necesario realizar la programación de esta clase a través del NDK de Android, algo que no es sencillo y que Google intenta que no se use en la medida de lo posible [31]. Debido a los consejos de Google, a la dificultad añadida y que en la mayoría de ocasiones el impacto en el rendimiento en imperceptible, esta aplicación se ha programado utilizando el API de Java.

3.3.3. Eclipse

Eclipse es lo que se denomina un IDE (Integrated Development Environment), es decir, es una aplicación que ofrece al programador una forma sencilla de desarrollar otras aplicaciones en un determinado lenguaje de programación. Suelen estar constituidas por un editor de código, herramientas de compilación y creación de la aplicación, un depurador y algún sistema de completado inteligente de código. El programa está escrito en su mayoría en Java, y fue diseñado principalmente para desarrollar aplicaciones en este lenguaje. No obstante, añadiendo los plugins necesarios, es capaz de crear aplicaciones en C, C++, JavaScript, Fortran, Lua, PHP, Phyton, entre otros muchos [32].

Este IDE de programación fue el elegido, en principio, para el desarrollo del proyecto. Para poder utilizarlo es necesario instalar en Eclipse un plugin llamado ADT, el SDK de Android, el SDK de OpenCV para Android y opcionalmente el NDK de Android. En su lugar Google recomienda en su página de Android Developers la descarga de Android Studio [33], pues desde hace un tiempo es el IDE oficial para trabajar en Android. El motivo de usar Eclipse + ADT es que hubo problemas en anteriores TFGs a la hora de utilizar OpenCV con Android Studio. Además el rendimiento de este último programa es claramente inferior al obtenido con Eclipse.

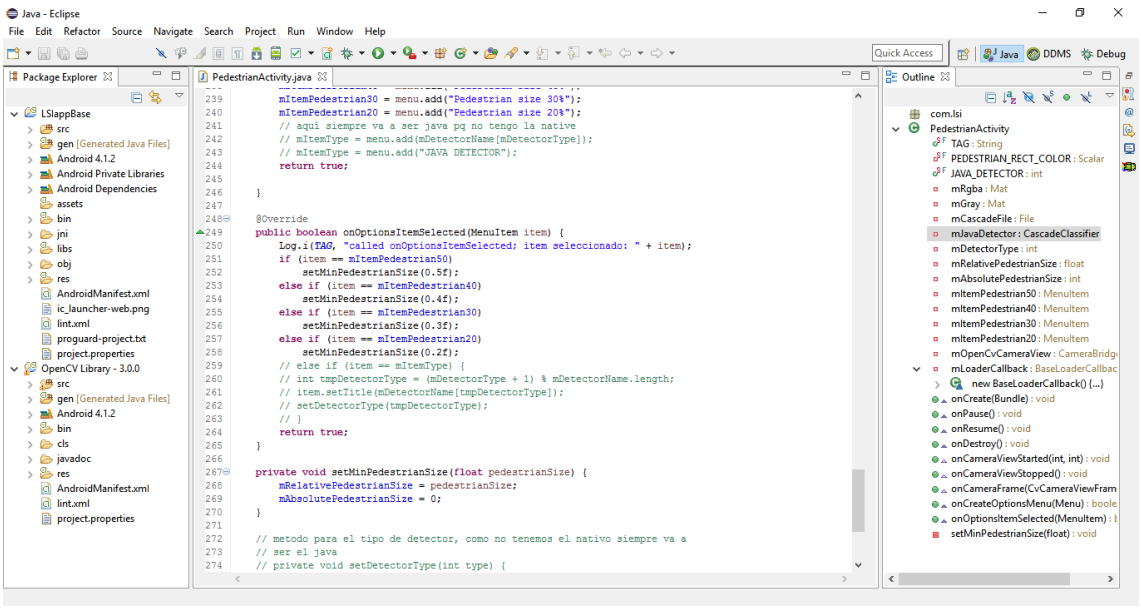


Figura 28: Vista principal del editor de código del IDE Eclipse

3.3.4. Android Studio

Android Studio, al igual que Eclipse, es un IDE, en este caso basado en la plataforma IntelliJ IDEA [34]. Esta plataforma de desarrollo está especialmente diseñada para trabajar con código de Java.

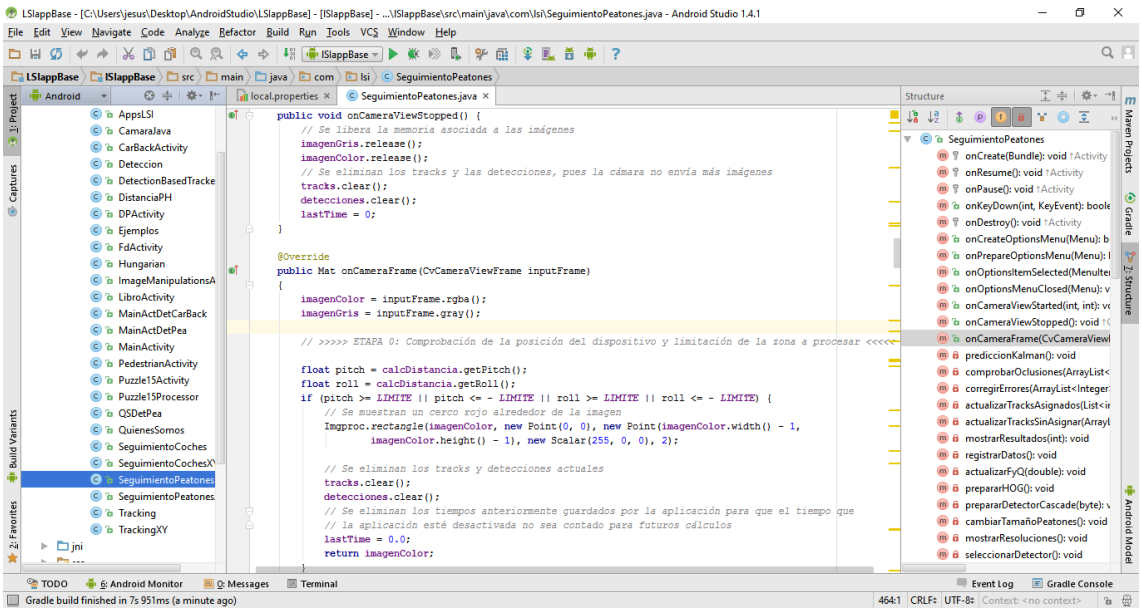


Figura 29: Vista del editor de código de Android Studio

A pesar de su menor rendimiento, en comparación con Eclipse, finalmente se portó la aplicación a este entorno, pues se ha convertido en el IDE oficial para el desarrollo de aplicaciones en Android y Google no puede asegurar que, si se sigue utilizando el plugin ADT, se

obtengan todas las nuevas características del SDK de Android. Android Studio incluye un rápido asistente que permite al usuario migrar un proyecto de ADT a este programa en tres sencillos clics, por lo que esto no supuso ningún problema. El problema en la migración consistía en que en la aplicación base existía un fragmento de código escrito en lenguaje nativo. La mayor parte estaba comentada, pues el autor de la aplicación base [71] explica que nunca fue capaz de que llegara a funcionar. Android Studio no soporta aun de forma oficial el NDK y además el código contenía errores, con lo que Android Studio no permitía la ejecución de la aplicación. Por tanto esa parte fue eliminada.

3.4. Contexto

En este apartado se estudiará el contexto de la aplicación, es decir, lo que la rodea, las bases que se han tomado para su desarrollo. Aquí se incluye por tanto la aplicación base en la que está recogida la desarrollada en este proyecto y el coche IVVI 2.0 del Laboratorio de Sistemas Inteligentes, pues sería el primer candidato en el que utilizar en la realidad la aplicación una vez que se incluyeran ciertas características que la completasen.

3.4.1. Aplicación base del LSI

Como ya se ha comentado, la aplicación aquí desarrollada está incluida dentro de la aplicación base que posee el Laboratorio de Sistemas Inteligentes. Fue diseñada por el estudiante del Grado en Ingeniería Informática Alejandro Ramos López [71] como proyecto de fin de grado. En ella se recogen todos los proyectos de fin de grado que va creando el laboratorio en el campo del uso de OpenCV para dispositivos Android. Actualmente incluye una serie de aplicaciones de ejemplo que trae consigo el SDK de OpenCV para Android, una aplicación que simplemente detecta peatones, otra que detecta coches, otra que permite realizar fotos y aplicar distintos kernels y por último las aquí diseñadas.

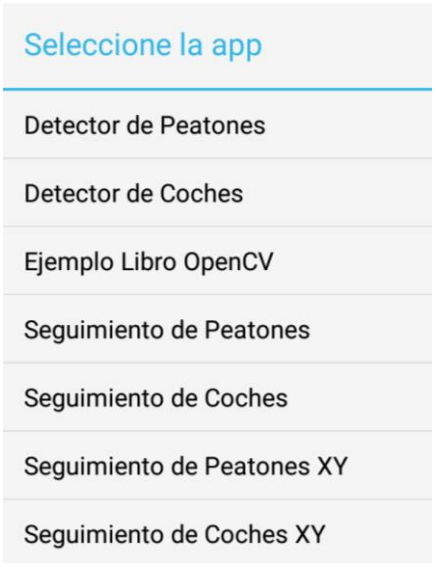


Figura 30: Aplicaciones disponibles en el app base de LSI

La idea es que en un futuro próximo se recojan todas las aplicaciones desarrolladas por otros alumnos en una sola versión de la aplicación base. Además muchas de las aplicaciones desarrolladas son una evolución de las anteriores, por lo que habría que mantener solo aquellas que ya incluyen todas las funciones posibles, para evitar la redundancia en la funcionalidad de las distintas aplicaciones.

3.4.2. Coche IVVI 2.0

Se trata de un vehículo usado por el Laboratorio de Sistemas Inteligentes en el cual investigadores y alumnos de la universidad prueban los sistemas ADAS que se van desarrollando a lo largo del tiempo en el LSI [35]. Estos sistemas están basados sobre todo en técnicas láser y en técnicas de visión por computador. En un futuro, cuando todas las aplicaciones de visión por computador para Android desarrolladas estén incluidas en la aplicación base, se podría probar su funcionamiento en este vehículo.

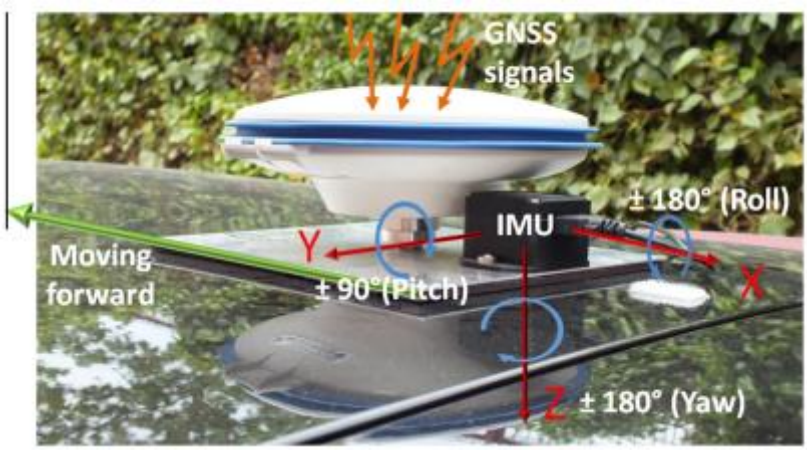


Figura 31: Unidades IMU y GNSS colocadas en el IVI 2.0 [35]

Los sistemas ADAS montados en el coche, así como los sensores utilizados para conseguir la funcionalidad deseada, son los que se enumeran a continuación:

- **Detección automática de señales de tráfico**: para ello se utiliza una cámara en color situada en la luna delantera.
- **Detección de obstáculos y espacio libre:** para ello se utiliza una cámara de visión stereo. Primero se calcula la profundidad de cada punto de la imagen, después se aplica un umbral para determinar si algo en la imagen es un obstáculo. Una vez identificados todos los obstáculos se eliminan todos los puntos que forman parte de ellos, quedando el mapa de espacio libre. Este mapa se procesa para obtener el perfil de la carretera.
- **Detección de peatones en el infrarrojo lejano:** utilizando una cámara de infrarrojos se puede llegar a identificar a los peatones de noche, como se explica en el apartado 2.3.1. de esta misma memoria.
- **Reconocimiento facial del estado del conductor:** el vehículo incluye una cámara Kinect de Microsoft con la que es capaz de, a partir de la imagen en color y de la información de distancia, monitorizar el estado del conductor. Esta información se procesa después para generar un sistema de alerta por distracción.

- **Detección y clasificación de obstáculos:** este sistema fusiona la información de un láser colocado en el parachoques delantero con las imágenes captadas por una cámara situada en el interior del vehículo. Esta fusión sensorial trae consigo una reducción importante del tiempo de procesamiento, así como un aumento de la fiabilidad del sistema. La detección de vehículos se realiza a través del detector Haar Cascade, mientras que la de peatones se realiza a través de HOG. Una vez han sido detectados se procede a realizar un tracking de los mismos a través de los algoritmos GNN (Global Nearest Neighbours) y UKF (Unscented Kalman Filter).
- **Sistema de posicionamiento en entornos urbanos:** para ello el IVVI 2.0 realiza una fusión sensorial entre GNNS e IMU. Como sistemas de navegación global por satélite se utilizan un GPS y un DGPS. Estos se fusionan con el sensor IMU a través del algoritmo UKF. El uso de fusión sensorial se debe a que el sistema GPS por sí solo presenta un error de unos 20 metros. Con el DGPS este error puede disminuir hasta un metro. No obstante, en entornos urbanos donde existen edificios altos o en zonas de la carretera como túneles y similares, la recepción de la señal GPS puede ser muy mala o incluso nula. Es aquí donde la unidad IMU entra principalmente en juego.

4. DESCRIPCIÓN DETALLADA DEL FUNCIONAMIENTO

En este apartado se va a proceder a explicar con más detalle el funcionamiento de la aplicación. En primer lugar se detallarán todos aquellos aspectos de la aplicación que están más relacionados con el propio entorno Android. Estos son básicamente aspectos visuales. Después se continuará con una explicación de en qué consisten ciertos algoritmos utilizados, como el Filtro de Kalman o el Algoritmo Húngaro, así como un repaso al sistema de cálculo de distancias. Finalmente se detallará la parte lógica de la aplicación, es decir, todos los pasos que han de seguirse para que esta realice su función.

4.1. Aspectos básicos de Android

Antes de comenzar a describir la aplicación visualmente es necesario introducir algunos conceptos importantes. Como ya se ha mencionado anteriormente, la app desarrollada está en realidad dividida en cuatro "subaplicaciones": detección y seguimiento de peatones en coordenadas de la imagen y en coordenadas reales y detección y seguimiento de coches en coordenadas de la imagen y en coordenadas reales. De forma más correcta se puede decir que está dividida en cuatro activities de Android. En el sistema operativo Android una activity es un componente de una aplicación que provee de una interfaz gráfica para interaccionar con el usuario [27]. Cada activity está formada por dos componentes: el layout o parte visual (escrita en archivos xml) y la parte lógica, escrita en lenguaje de programación Java (archivo con formato .java en el que se recogen la clase o clases necesarias). Todos los componentes visuales del layout están enlazados con la parte lógica, pues es allí donde se programa su comportamiento. En lo que resta de memoria, cuando aparezca la palabra aplicación, en realidad se está refiriendo a una activity, a excepción de cuando se nombra a la aplicación base.

En Android una aplicación está formada por una o varias activities. En el caso de la aplicación base la activity principal es el menú principal. Todos los demás menús y pantallas existentes son otras activities distintas.

Para crear una activity en Android simplemente hay que hacer que una clase herede de la clase Activity, lo cual implica tener que implementar algunas de las funciones que controlan el ciclo de vida de una activity en Android. Así mismo, para poder utilizar la librería de OpenCV, es necesario implementar la clase CvCameraViewListener2 en una activity.

Es importante indicar que en cualquier aplicación de Android se pueden ver al menos tres elementos visuales en forma de barra: el Status Bar, el Action Bar y el Navigation Bar. El Status Bar o barra de estado es la que se encuentra en la parte superior de la pantalla. Es el lugar en el que aparecen las notificaciones de Android y otros símbolos como el nivel de cobertura o de batería. La segunda barra es el Action Bar o barra de acciones. Esta barra es la que aparece justo debajo de la barra de estado. En ella suele aparecer el símbolo de la aplicación, el nombre de la activity y el menú de opciones. Por último tenemos la Navigation Bar o barra de navegación. Esta se sitúa en la parte inferior de la pantalla y recoge los botones de casa o principal, atrás y vista de aplicaciones o menú (solo para dispositivos antiguos). En muchos dispositivos esta última no es una barra como tal, sino que los botones son físicos.

Figura 32: Status Bar y Action Bar (izquierda) [65]. Navigation Bar (derecha) [66]

4.2. Layout de la aplicación

El layout (o parte visual) de cada una de las cuatro activities es idéntico, simplemente consta de un elemento del tipo CamaraJava que ocupa toda la pantalla. Este deriva de la clase JavaCameraView, la cual está incluida en OpenCV y su función es la de actuar de puente entre la propia librería OpenCV y la cámara de Android. Todas las opciones de la aplicación son accesibles a través del menú de la misma; no existen elementos visuales que dificulten la visión de la ejecución de la aplicación.

En general la pantalla de un smartphone no suele ser muy grande, alrededor de las 5 pulgadas. Si a esto le sumamos las barras de estado, navegación y de acciones, la imagen visualizada en la pantalla se hace demasiado pequeña. Por ello, y para maximizar el tamaño en pantalla de los frames que la cámara está devolviendo, se ha optado por ocultar de forma predeterminada todas estas barras.

Cuando una de las aplicaciones se abre aparece a pantalla completa, sin ninguna barra visible. El problema es que el usuario no puede acceder al menú de la aplicación. Para solucionar esto se ha implementado la interfaz OnTouchListener en cada activity, lo que permite a la aplicación reaccionar ante los toques de pantalla. Cuando el usuario toca la pantalla y las barras están ocultas, estas se vuelven a mostrar. Si el usuario vuelve a tocar la pantalla, estas se ocultan de nuevo.

Sin embargo existe un problema en el proceso anterior. De forma predeterminada Android coloca las distintas barras en su lugar ocupando espacio físico en pantalla. El layout de la aplicación se dibuja entonces en el espacio disponible entre barras. Si en esta circunstancia intentamos ocultar todas estas barras, Android tiene que redimensionar el layout de modo que ocupe todo el espacio libre (ahora toda la pantalla). Esta no es una tarea trivial, pues tiene que eliminar el layout anterior y redibujarlo a pantalla completa. Este proceso requiere de una gran cantidad de recursos, con lo que la aparición y desaparición de las barras resulta en una animación poco suave y muy poco agradable a la vista. Para evitarlo hay que navegar por el árbol del proyecto hasta res/values/styles/styles.xml y añadir dentro del espacio reservado para el tema de la aplicación la siguiente línea:

```
<item name="android:windowActionBarOverlay">true</item>
```

Con esto conseguimos que el layout ocupe toda la pantalla y que las distintas barras del sistema Android se dibujen por encima. De esta forma, cuando se ocultan o se muestran, el

layout no tiene que ser redibujado, lo que produce transiciones suaves. El tema de una aplicación, anteriormente nombrado, está descrito en el archivo styles.xml y determina el aspecto visual de todos los elementos de la interfaz gráfica como los widgets o el color del menú.

El aspecto visual de la aplicación de detección de peatones puede verse en la figura 4.

4.3. Aspectos básicos de la aplicación

Antes de entrar más en profundidad en el funcionamiento de la aplicación es necesario aclarar algunos conceptos que se han ido mencionando a lo largo de esta memoria.

Recordemos que la aplicación desarrollada está dividida en 4 partes. Dos de ellas realizan el seguimiento en coordenadas de la imagen (U y V), para peatones y para coches, mientras que las otras dos hacen lo propio en coordenadas reales X e Y. A continuación se muestran imágenes que visualizan el sistema de coordenadas utilizado en los dos casos.

Además del sistema de coordenadas utilizado en cada caso, también se puede ver la posición del punto de interés, la forma que tiene la aplicación de indicarnos donde ha encontrado a un peatón y la distancia a la que se encuentra, así como la delimitación del área donde se buscan objetos.

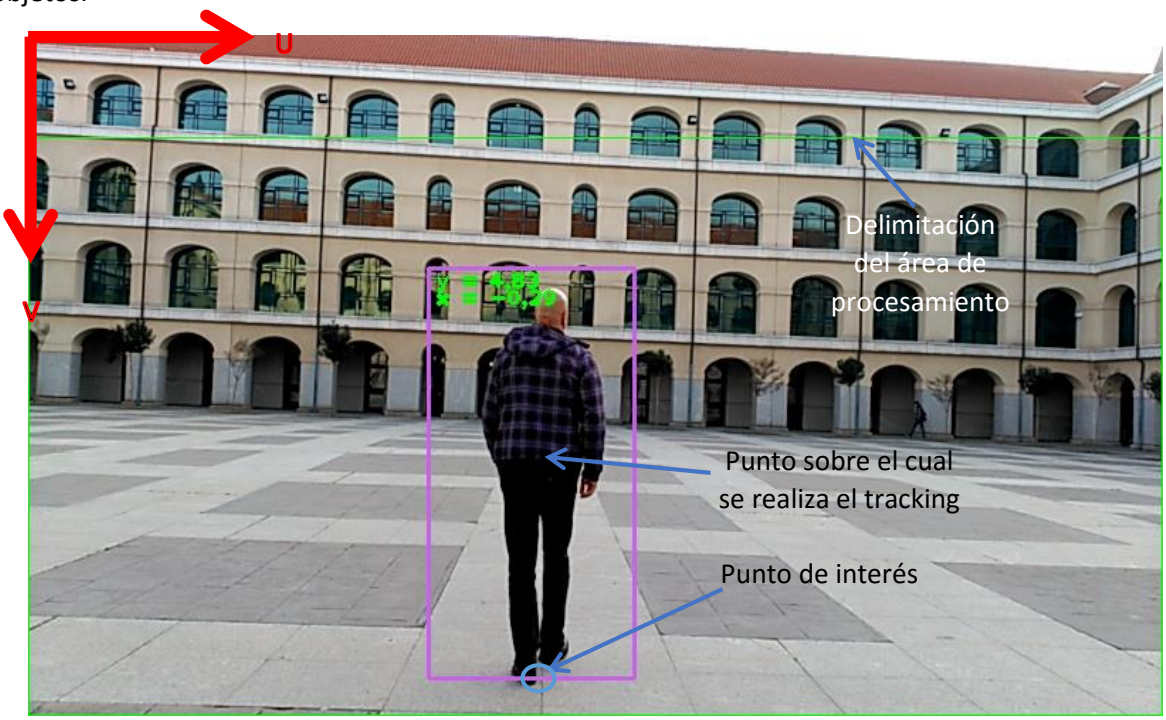


Figura 33: Sistema de coordenadas de la imagen

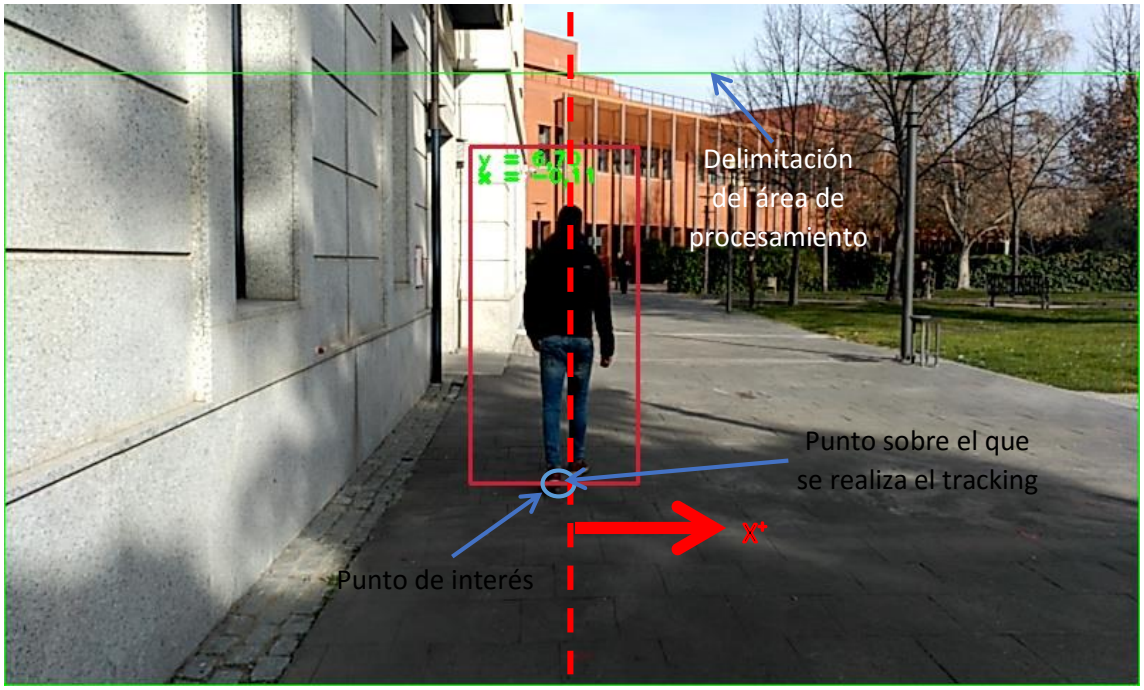


Figura 34: Sistema de coordenadas reales. La distancia Y representa la profundidad a la que se encuentra el objeto, por tanto su eje tiene su origen en la cámara del dispositivo y está orientado hacia el objeto, paralelo al suelo.

4.4. Menú de opciones de la aplicación

Los distintos elementos del menú de opciones, así como su función, se detallan a continuación:

- **Cambiar resolución**: la resolución de los frames devueltos por la cámara se puede cambiar a través de este menú. Para ello se muestra una lista que contiene todas las resoluciones que admite la cámara.
- **Seleccionar altura**: el sistema de estimación de distancias utilizado necesita conocer la altura a la que se encuentra el dispositivo móvil respecto del suelo. Para ello se muestra un diálogo en el que el usuario puede introducir esta altura en metros.
- **Modificar margen**: cuando el dispositivo móvil se coloca en la luna delantera del coche en posición horizontal, normalmente existe un margen superior en el que no se espera que aparezca ningún peatón o coche. Los peatones normalmente aparecerán y desaparecerán por la izquierda o la derecha. Por su parte los coches aparecerán también por un lateral y desaparecen por el horizonte, sobre la mitad de la pantalla. Por todo esto, el usuario puede elegir un margen superior en el cual no se realiza ningún tipo de detección. Además, al disminuir el tamaño de la imagen a procesar, conseguimos disminuir la carga de procesamiento y aumentamos los fps. El valor por defecto es un 15%.
- **Tamaño de peatones/coches**: cuando se usa el detector Haar Cascade necesitamos indicarle el tamaño en píxeles mínimo de los objetos a buscar. Cualquier objeto cuyo tamaño sea menor es ignorado. En el caso de estar utilizando HOG este menú se desactiva automáticamente. El tamaño por defecto es un 20 % del tamaño total.

- **Seleccionar detector**: en el caso de la detección de peatones permite cambiar entre los detectores Haar Cascade y HOG. En el caso de la detección de coches este menú no está disponible, pues solo se utiliza Haar Cascade.
- **Salir**: la aplicación guarda todos los parámetros anteriores y se cierra.

Como puede verse en el apartado dedicado al sistema operativo Android, la primera función a la que se llama cuando se inicia una activity es a onCreate(). En ella se suelen realizar tareas iniciales como dar un valor a los atributos de la clase. En este caso además se utiliza para llamar a una función que recupera todos los datos anteriormente mencionados. Los datos guardados por las dos aplicaciones de detección de peatones se guardan aparte de los mismos utilizados en la detección de coches. Además cada detector guarda por separado un valor para el margen y para la resolución, puesto que no tienen por qué ser iguales para ambos.

El guardado de los datos se realiza en las llamadas a las funciones onPause y onDestroy. onPause es llamada por el sistema operativo cuando la aplicación deja de estar en primer plano, mientras que onDestroy es llamada cuando el usuario decide cerrar la aplicación o la misma está siendo destruida por Android por falta de recursos.

4.5. Algoritmos de detección

Como algoritmos de detección para peatones se han utilizado Haar Cascade [36] y HOG [18]. Para la detección de coches solo se ha usado Haar Cascade. A continuación se explicará brevemente en que consiste cada uno de ellos y su implementación en la aplicación.

- **Haar Cascade**

Fue diseñado en 2001 por Paul Viola y Michael Jones con el objetivo de crear un detector de caras extremadamente rápido y robusto. Para detectar los objetos deseados el clasificador utiliza características de tipo Haar, las cuales se pueden observar a continuación:

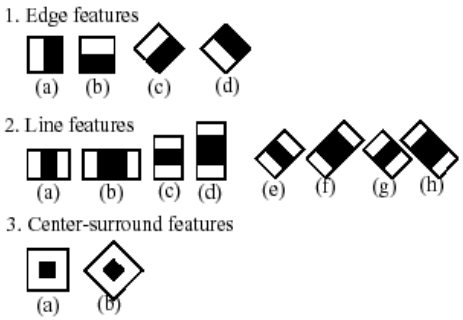


Figura 35: Características tipo Haar [37]

El procedimiento consiste en mover por toda la imagen una ventana de búsqueda, en la cual se aplica el clasificador, el cual a su vez devuelve '1' si dicha región de la imagen es susceptible de contener el objeto buscado, o '0' en caso contrario. El clasificador toma una de las anteriores características Haar y la aplica sobre la imagen a modo de máscara. Todos los píxeles que queden bajo la parte blanca se suman (sus niveles de gris), mientras que los que queden bajo la zona negra restan. Con esto obtenemos un valor, que si es el esperado, significará que hay

posibilidades de haber encontrado el objeto buscado. El término Cascade hace referencia a que no se utiliza un solo clasificador, sino que se utilizan varios clasificadores sobre una misma región de forma secuencial. Esto significa que los clasificadores están formados por árboles de decisión. Si en cualquier momento de la secuencia el valor no es el esperado para cualquiera de los clasificadores, se entiende que en esa región no se encuentra el objeto buscado e inmediatamente se pasa a la siguiente región.

En cuanto a OpenCV, la función que nos permite detectar objetos mediante Haar Cascade se llama detectMultiScale (de la clase CascadeClassifier) y necesita los siguientes parámetros:

- **Image**: imagen en escala de grises donde realizar la búsqueda.
- **Objects**: es una lista de rectángulos donde cada rectángulo contiene un objeto deseado, en este caso peatones o coches. Este es por tanto el resultado de la función y de la detección en sí.
- **scaleFactor**: para poder detectar objetos de diferentes tamaños en una misma imagen esta debe ser redimensionada varias veces, en cada una de las cuales se realizará una búsqueda del objeto deseado. Este parámetro especifica cuanto ha de reducirse la imagen para cada uno de estos redimensionamientos. El valor utilizado en todas las aplicaciones es 1.1.
- **minNeighbors**: cuando se detecta un objeto, la función no tiene por qué devolver en principio un único rectángulo que lo contiene. En su lugar se generan una serie de rectángulos alrededor de la zona donde se encuentra el objeto. Este parámetro especifica cuantos rectángulos tiene que tener uno dado a su alrededor para considerarlo válido. Aumentando este valor conseguimos estar más seguros de que hemos detectado lo que queremos, pero habrá muchos objetos que se queden sin detectar. Si este valor baja mucho, se obtendría una gran cantidad de falsos positivos. Cuando se ha terminado la detección, el algoritmo se encarga de agrupar todos los rectángulos de una misma región en uno solo, el promedio de todos ellos. El valor utilizado es 3.
- **Flags**: solo utilizado en las versiones 1.x de OpenCV.
- **minSize**: tamaño mínimo de los objetos a detectar. Cualquier objeto menor es descartado de forma automática. El valor por defecto es un cuadrado cuyo lado equivale al 20 % del tamaño en horizontal de la imagen.
- **maxSize**: tamaño máximo del objeto a detectar. Cualquier objeto mayor es descartado. En la aplicación no se especifica un tamaño máximo.

- **HOG**

La idea que hay detrás de este método es que la forma y apariencia de un objeto pueden ser caracterizadas por la distribución de la intensidad de los gradientes o de la orientación de los mismos [18]. Un gradiente en una imagen representa un cambio en la intensidad de gris, o lo que es lo mismo, un borde. Para obtenerlos es necesario calcular el gradiente de la imagen. Dicha imagen se divide en pequeñas regiones llamadas celdas, y para cada celda se calcula un histograma de una dimensión de la orientación de los gradientes en dicha celda. Esto quiere decir que para cada celda se calcula una representación de la frecuencia con la que aparecen las distintas orientaciones de los gradientes de la celda. Para hacer el algoritmo más robusto a cambios de iluminación es recomendable normalizar el contraste. Para ello se genera un bloque

mayor que contenga varias celdas y se utiliza la información de todas ellas para normalizar los valores de cada una. Cada uno de los descriptores HOG obtenidos de cada bloque se combina en un vector de características que posteriormente se lleva a un SVM (Support Vector Machine), el cual decide si hay o no una persona. El proceso detallado se puede consultar en [18].

En OpenCV la función que detecta peatones haciendo uso de descriptores HOG también se denomina detectMultiScale (perteneciente a la clase HOGDescriptor) y contiene los siguientes parámetros:

- **Img:** imagen en la que buscar a los peatones. Puede ser en escala de grises o en color. En la aplicación se utiliza en escala de grises porque requiere menor capacidad de procesamiento.
- **foundLocations**: lista de rectángulos que contiene los peatones detectados. Este es por tanto el resultado principal de la función.
- **foundWeights:** lista de coeficientes que indican la seguridad del algoritmo de haber encontrado realmente un peatón.
- **hitThreshold**: distancia mínima entre el plano del SVM y las características. Debe tener un valor de cero.
- **winStride:** número de pixeles o paso que ha de avanzar cada vez la ventana de detección por la imagen.
- **padding:** debe tener valor 0.
- **scale:** al igual que ocurre con el detector Haar Cascade, para detectar peatones de distintos tamaños es necesario redimensionar la imagen y realizar la búsqueda en todas ellas. Según [18] un valor de 1.05 es el más apropiado.
- **finalThreshold:** similar al atributo **minNeighbors** en el detector Haar Cascade.
- **useMeanShiftGrouping:** el algoritmo utiliza un método más complejo que el utilizado a través de finalThreshold que incluye el grado de seguridad de detección para agrupar todos los rectángulos similares en uno.

Es necesario destacar que este último algoritmo presenta una tasa de detección mayor que Haar Cascade [44], sin embargo, requiere de mayor capacidad de cómputo. Para poder aplicarlo a este proyecto se ha tenido que disminuir la resolución de las imágenes tomadas por la cámara, en comparación con Haar Cascade. Esto no es un problema pues este algoritmo se basa en calcular la dirección de los gradientes en la imagen, la cual no cambia al reducir su tamaño.

4.6. Filtro de Kalman

El filtro de Kalman es un algoritmo que se utiliza para estimar el estado de un sistema dinámico lineal cuando dicho sistema está sometido a ruido blanco aditivo. La idea que se aplica a este proyecto es que, si el movimiento de un objeto es continuo, sería posible realizar predicciones sobre él en cada instante basándonos en trayectorias previas.

El ruido blanco es una señal aleatoria que se caracteriza porque sus valores de señal en dos instantes de tiempo diferentes no guardan correlación estadística. Esto significa que el valor del ruido blanco en un determinado instante de tiempo no depende del valor de la señal en instantes anteriores.

El funcionamiento del filtro se basa en calcular en un determinado instante de tiempo una estimación del estado del sistema a partir de la estimación del estado anterior y las entradas al sistema. Esto se realiza durante la llamada etapa de estimación. A este este estado estimado se le denomina estimación a priori. Posteriormente llega una medida procedente de uno o varios sensores que recoge el valor de uno o varios estados, con ruido blanco incluido. A partir de esta medida ruidosa el filtro es capaz de corregir la estimación anterior. A este estado estimado se le denomina estimación a posteriori. Las ecuaciones que rigen este funcionamiento son las siguientes:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k$$
$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k$$

Figura 36: Etapa de estimación del filtro de Kalman [39]

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}$$
$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k$$
$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$$
$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$$
$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}$$

Figura 37: Etapa de actualización del filtro de Kalman [39]

El filtro de Kalman no está obligado a pasar siempre por las fases de predicción o estimación y actualización. Puede ser que solo pase por una de ellas. En ese caso lo más común es que solo pase por la fase de predicción. Esto ocurre cuando en un determinado instante k no hay disponible una medida para corregir la estimación. En un sistema de tracking o seguimiento esto ocurre cada vez que el sistema pierde un track, por ejemplo porque está ocluido. En ese caso solo se predice su posición, por lo que la incertidumbre de su estado aumenta con el paso del tiempo. Cuando esta incertidumbre supera un cierto límite el track es eliminado.

- **Modelo del sistema**

El sistema físico se modeliza por un vector de estados **x** (de n componentes, donde n es el número de estados) y un conjunto de ecuaciones llamado modelo del sistema, el cual describe la evolución del estado con el tiempo.

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \quad \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)$$

Figura 38: Ecuación del modelo del sistema [39]

F_k es una matriz de n x n llamada matriz de transición de estados. Describe como se obtiene el estado en el instante k a partir del instante k-1. B_k es la matriz de control que se aplica al vector de entradas al sistema u_k. En el caso de la aplicación aquí desarrollada no existen entradas al sistema, solo medidas a través de la cámara, por lo que B_k es cero. w_k es un vector que modeliza el ruido asociado al modelo del sistema, es decir, modeliza como de cercano es el modelo al sistema real. w_k está representada por una distribución normal de media cero y covarianza Q_k. A esta última matriz se la denomina matriz de covarianza del error de modelo.

El estado del sistema (compuesto de variables del sistema) está representado en el filtro de Kalman por una variable gaussiana de media el vector de estados **x** y una matriz de covarianzas P_k. A la matriz P_k se la denomina matriz de covarianzas del error de estado. Cuanto mayor sea esta matriz, más error tendrá la estimación, y por tanto, más alejada estará del estado real del sistema. El valor de esta matriz es muy importante, ya que muchos sistemas de seguimiento que utilizan este filtro se basan en el valor que toma para saber cuándo eliminar a un track.

- **Modelo de medida**

Para cada instante k el sistema recibe una observación ruidosa del vector de estados, o de alguna de sus componentes.

$$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \quad \mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)$$

Figura 39: Ecuación del modelo de medida [39]

z_k es el vector de medidas tomadas en el instante k. H_k es la llamada matriz de medidas y realiza el mapeo entre los estados observados y los estados reales o estimados. v_k es un vector aleatorio que modeliza la incertidumbre asociada a las medidas. Está definido a partir de una gaussiana de media cero y una matriz de covarianzas R_k. R_k es la llamada matriz de covarianza del error de medida de los estados del sistema (errores de los sensores de medida, que en este caso se deben a errores en la posición en la que se detectan peatones y coches y a errores en la estimación de las distancias).

El filtro de Kalman es un estimador recursivo, lo que significa que solo necesita conocer el estado estimado del sistema en el instante anterior K - 1 y el vector de medidas en el instante k para calcular la estimación del estado en el instante k [39]. Esto implica que no necesita recordar ningún historial de estados o medidas pasados y por tanto resulta muy interesante su uso para aplicaciones en tiempo real.

Una de las características más interesantes de este filtro es que permite realizar un suavizado del estado del sistema. El filtro cuenta con que las medidas que llegan al sistema no son fiables, pues contienen ruido blanco. Por ello, en la etapa de actualización, cuando se calcula la estimación del estado a posteriori, dicha estimación no se actualiza directamente con la medida recibida. En lugar de ello se calcula la diferencia entre la medida observada y la estimación de esta medida a partir de los datos de la etapa de predicción. La estimación a posteriori se calcula entonces a partir de la estimación a priori, la ganancia de Kalman K_k y la diferencia anteriormente calculada. Cuanto mayor sea el valor de la matriz R_k menos fiables serán las medidas, y por tanto, menos corrección aplicará el filtro, con lo que el resultado estará más suavizado.

El concepto de suavizado explicado en el anterior párrafo es especialmente importante en este proyecto, pues se está utilizando un filtro de Kalman lineal (o simplemente filtro de Kalman) con un modelo del sistema de velocidad constante. Esto es una aproximación, pues el movimiento de los peatones y de los coches no es de velocidad constante, existen aceleraciones. No obstante cabe destacar dos hechos que hacen que esta aproximación sea válida:

- Los frames por segundo a los que trabaja el algoritmo son suficientemente altos como para que las no-linealidades entre dos frames consecutivos sean muy pequeñas.

- La matriz Q del sistema, como ya se ha dicho, se utiliza en cierto modo para modelar las imperfecciones del modelo. A medida que aumenta Q el algoritmo "entiende" que el modelo está más alejado de la realidad. Escogiendo un modelo adecuado de Q podemos incluir en el sistema el efecto de las aceleraciones.

Siguiendo con todo lo anterior, no podemos olvidar que existen dos formas de realizar el seguimiento de peatones y coches: mediante coordenadas de la imagen (píxeles de la imagen) y mediante coordenadas reales (profundidad y desviación respecto del centro de la imagen en metros). Cuando se utilizan coordenadas reales es necesario tener en cuenta que estas son una estimación, por lo que las distancias medidas presentan un error que puede constituir una fuente más de no-linealidades. Por tanto, en teoría, las aplicaciones que utilizan coordenadas reales deberían presentar mayores errores de seguimiento.

En el apartado 4.10 se pueden observar los valores concretos que toman las matrices del filtro de Kalman durante la ejecución de la aplicación.

4.7. Algoritmo Húngaro

Uno de los primeros problemas que aparecieron al desarrollar el proyecto fue el siguiente: una vez que el detector utilizado analiza la imagen y encuentra los objetos a seguir (peatones o coches) es necesario asociar de alguna forma los nuevos objetos (encontrados en el frame actual) con aquellos que ya estaban presentes en frames anteriores, para después poder informar al filtro de Kalman de la nueva posición de cada uno de los anteriores tracks (objetos en seguimiento). Es decir, para cada frame, hay que decidir para cada objeto detectado si es un nuevo objeto, o si por el contrario ya estaba presente en frames anteriores. De ser este el caso, hay que calcular a que objeto en seguimiento (track) pertenece la nueva detección. En definitiva con este proceso lo que se está haciendo es calcular el movimiento de los tracks entre frames consecutivos. A continuación se presenta un pequeño esquema del problema sobre el que se aplica el Algoritmo Húngaro.

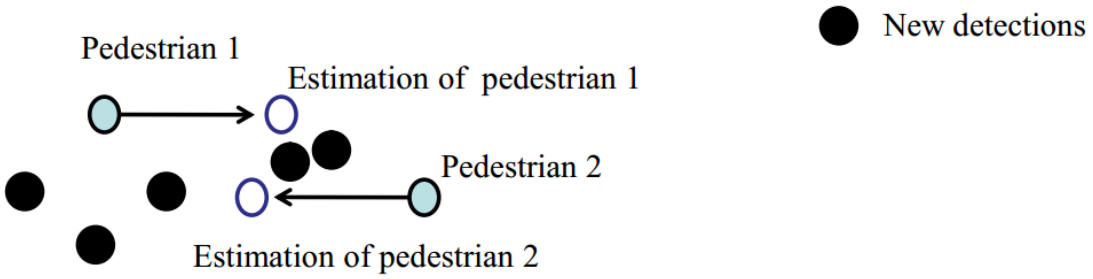


Figura 40: Problema de asociación

Como puede observarse en la imagen, las estimaciones del filtro de Kalman sobre las nuevas posiciones de los tracks en seguimiento no tienen por qué coincidir con la posición de los objetos detectados. Es el Algoritmo Húngaro quien asocia cada peatón o coche en seguimiento con una nueva detección, siempre que sea posible.

Para evitar asociaciones absurdas y simplificar el proceso se utilizan técnicas de gating, lo que significa que existe una distancia máxima entre una estimación y una detección para que estas se puedan asociar. Los cálculos detallados se encuentran en el apartado 4.10.

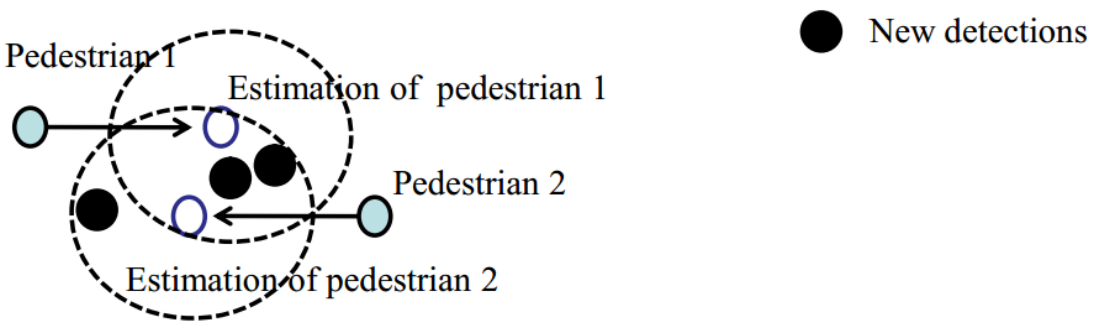


Figura 41: Aplicación del gating

El Algoritmo o Método húngaro es un algoritmo diseñado para encontrar una asignación óptima dada una matriz de costes. Por asignación óptima se entiende aquella que minimiza el coste total [40]. Como caso general, una matriz de coste se puede generar representando en cada fila un recurso y en cada columna una tarea a realizar. Cada celda representará entonces el coste de asociar un determinado recurso con la realización de una determinada tarea. Solo se puede asociar un recurso a cada tarea y viceversa. Es decir, no puede haber un recurso con dos tareas ni una tarea asociada con dos recursos.

La aplicación del Método Húngaro en este proyecto consiste en crear una matriz de costes en la que cada fila representa la predicción de cada track en un determinado frame. Cada columna representa cada una de las nuevas detecciones que los algoritmos de detección han encontrado. Por último, cada celda de la matriz contiene la distancia entre cada track y cada detección nueva. Al aplicar este método se consigue emparejar a cada track con la detección más cercana, o lo que es lo mismo, se empareja cada detección con el track más cercano a ella.

$$\begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \end{array} \begin{array}{c} \begin{array}{ccc} d_1 & d_2 & d_3 \end{array} \\ \begin{bmatrix} 0{,}2 & 1{,}8 & 2{,}6 \\ 1 & 2 & 0{,}8 \\ 1{,}5 & 0{,}3 & 1 \end{bmatrix} \end{array}$$

En el ejemplo superior se representa la salida del Algoritmo Húngaro dado el caso en el que se tuvieran tres tracks procedentes de imágenes anteriores y tres nuevas detecciones. El detalle de los pasos que sigue el Método Húngaro para llegar a este tipo de solución puede consultarse en [40].

Es evidente que puede darse el caso en el que existan más detecciones nuevas que tracks (en el frame actual aparecen nuevos objetos) o que existan más tracks que detecciones (caso que se da por ejemplo cuando un objeto está ocluido por otro). El procedimiento a seguir en estos casos se detalla en el apartado 4.10, pero básicamente consiste en crear los tracks o detecciones fantasma (irreales) necesarios para hacer que la matriz de costes sea cuadrada (requisito necesario para que el algoritmo funcione). Las celdas asociadas a estas filas o columnas se rellenan con un valor lo más alto posible, de forma tal que, al tener asociado un coste tan alto, no influyan a la asociación del resto de elementos. Como la matriz es cuadrada y se han de

asociar siempre los elementos de las filas con los elementos de las columnas, habrá detecciones o tracks fantasma asociados a otros objetos reales. Cuando un track real se asocia a una detección fantasma significa que el track no está visible en el frame actual, pues la detección no existe realmente. En el caso de que una detección real se asocie con un track que realmente no existe, significará que estamos ante un posible nuevo objeto a seguir.

4.8. Sistema de estimación de distancias

Este sistema fue desarrollado por Víctor Cancho Armesto como trabajo de fin de grado [70]. En concreto el trabajo se titula "biblioteca para localización de peatones en dispositivos Android". En este trabajo de detalla el procedimiento a seguir para poder calcular una estimación de la distancia a la que se encuentra cualquier tipo de objeto, con una sola cámara, a partir de un punto de interés situado en el plano del suelo. Si el dispositivo móvil se encuentra paralelo al objeto, o que es lo mismo, la cámara apunta de forma paralela al plano del suelo, la distancia puede obtenerse a través del clásico modelo Pin-Hole. No obstante, una de las características más importantes de ese proyecto es que permite realizar esa mima estimación cuando el smartphone está inclinado hacia arriba o hacia abajo un ángulo φ, llamado "pitch" en el referido trabajo.

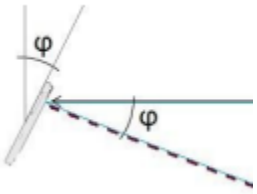


Figura 42: Ángulo de inclinación pitch con el dispositivo tumbado y de perfil

Para poder tener en cuenta el ángulo de inclinación el autor del anteriormente mencionado TFG desarrolló tres métodos. El primero fue extraído del libro publicado por Arturo de la Escalera [41], el segundo se basa en la aplicación de las matrices de rotación de Euler, mientras que el tercero fue una obra personal del propio autor del TFG. En este proyecto solo se utiliza por defecto el primer método, pues es el que mejores resultados ofrece, según indica el propio autor y según se ha podido comprobar. No obstante, las funciones necesarias para realizar esta compensación mediante los otros métodos están también incluidas en la aplicación y se pueden llegar a utilizar si se desea.

En el apartado 3.1 de esta memoria se especifican además una serie de mejoras que se aplicaron sobre esta librería cuando se añadió al presente proyecto.

Para estimar la distancia entre el dispositivo y el peatón o coche detectado, esta biblioteca necesita tomar el punto de contacto entre el plano del suelo y el objeto como punto de interés. En la práctica este punto de interés se toma como el punto central del borde inferior del rectángulo que envuelve al objeto detectado. Evidentemente los algoritmos de detección utilizados no son perfectos, y los rectángulos envolventes que devuelven no se adaptan a la perfección al objeto detectado, por lo que esto supone una fuente de error. En general este

rectángulo es más grande que el objeto real, por lo que es necesario adaptarlo, tal y como se detalla en el apartado 4.10 de esta memoria.

Un punto importante a destacar es que esta biblioteca se basa en un modelo de suelo plano. Esto implica que si esta condición no se da, la distancia estimada presentará un error. Dado que las vías de circulación no suelen presentar grandes cambios de pendiente, esto no debería de suponer una fuente de error importante, puesto que el suelo deja de ser plano en los cambios de pendiente, no una vez estamos dentro de la propia pendiente. Si la pendiente es descendente los objetos aparecerán más abajo en la imagen, con lo que la distancia calculada será menor que la real. Lo contrario ocurre en el caso de pendientes ascendentes.

También es necesario recalcar que el cálculo de la inclinación del dispositivo con respecto al suelo está basado en el cálculo del vector de gravedad a través del acelerómetro del dispositivo móvil. Si el vehículo sobre el que va montado el dispositivo entra en una zona de pendiente constante, desde el punto de vista del vehículo y del dispositivo, el suelo sigue siendo plano y el ángulo que forma el dispositivo con el suelo se mantiene constante, por lo que no debería de existir ningún tipo de error en ese aspecto. No obstante, al entrar en la zona de pendiente, el vector de gravedad cambia, por lo que la aplicación determina que el dispositivo ha sido inclinado y que el ángulo que forma el dispositivo con el suelo ha cambiado, con lo que se produce un error en la estimación de la distancia.

Para concluir este apartado solo queda indicar que el algoritmo de estimación de distancias basa su funcionamiento en calcular la diferencia en píxeles entre el centro de la imagen y el punto de interés en la coordenada V. Se escoge el centro de la imagen porque este punto es el único que no presenta distorsión causada por la lente. En realidad el punto que no presenta distorsión no tiene por qué coincidir exactamente con el centro de la imagen, sin embargo, es lo más común y además los fabricantes de dispositivos móviles no detallan este parámetro.

4.9. Clases implementadas

El objetivo de este apartado es enumerar las distintas clases que han sido desarrolladas en el proyecto, así como incluir un pequeño resumen de la función del código que contienen:

- **CamaraJava**: se trata de un clase que deriva de la clase JavaCameraView. Esta última está implementada en la librería de OpenCV y su función es la de servir de puente entre la propia librería OpenCV y la cámara del dispositivo. CamaraJava se encarga de realizar todas las operaciones que tienen relación directa con la cámara. Esto es, obtiene y cambia la resolución actual de los frames de la cámara y contiene la lógica que permite guardar capturas de pantalla de lo que ocurre en la aplicación.
- **Deteccion**: representa a los objetos detectados. Incluye información sobre su posición en coordenadas reales X e Y, así como el recuadro o bounding box que encierra al objeto detectado, representado en coordenadas de la imagen.
- **DistanciaPH**: en esta clase está implementada la biblioteca de estimación de distancias.
- **Hungarian**: implementación del Algoritmo Húngaro.

- **SeguimientoCoches**: como su nombre indica, contiene toda la lógica necesaria para realizar la detección y seguimiento de los coches. El seguimiento se realiza en coordenadas de la imagen (coordenadas U y V, en píxeles).
- **SeguimientoCochesXY**: realiza las mismas acciones que la clase anterior, pero en este caso en coordenadas reales.
- **SeguimientoPeatones**: contiene la lógica necesaria para realizar la detección y seguimiento de peatones en coordenadas de la imagen.
- **SeguimientoPeatonesXY**: igual que en el caso anterior, pero para coordenadas reales.
- **Tracking**: representa a los tracks en seguimiento, ya sean coches o peatones. Incluye por ejemplo el filtro de Kalman asociado a cada track y agrupa diversas funciones encargadas de calcular la distancia o el grado de solapamiento entre tracks o entre tracks y detecciones (necesario para que el Algoritmo Húngaro determine las asociaciones entre tracks ya en seguimiento en frames anteriores y las detecciones encontradas en el frame actual).
- **TrackingXY**: realiza las mismas funciones que la clase anterior, solamente que esta es utilizada por aquellas aplicaciones que trabajan sobre coordenadas reales. Ambas clases están separadas porque las matrices que forman el filtro de Kalman no se inicializan de la misma forma y porque desde un principio fueron desarrolladas de forma independiente.

En estas dos últimas clases cabe destacar que los parámetros más importantes que identifican a un track son el filtro de Kalman asociado, la predicción de la posición en el frame actual, el track propiamente dicho (bounding box más distancias en los ejes X e Y reales para el estado actualizado, si existe en el frame actual) y las tres últimas detecciones asociadas al track, incluida la del frame actual, si existe.

Podría parecer, por sus semejanzas, que se podrían haber unido las clases SeguimientoCoches y SeguimientoCochesXY, así como SeguimientoPeatones y SeguimientoPeatonesXY, pero cada una de estas clases se desarrolló como una activity o "aplicación" independiente, por lo que deben permanecer separadas. Además esto obligaría a llenar el código de sentencias condicionantes que dificultarían su lectura y comprensión.

4.10. Funcionamiento de la aplicación

El ciclo de funcionamiento de todas las aplicaciones desarrolladas (cuatro en total) es muy similar, de tal forma que se pueden resumir todas en uno mismo. Si en un momento determinado esta circunstancia no se diera, esto será aclarado convenientemente.

El funcionamiento está basado en repetir, para cada frame, una secuencia de 8 o 9 etapas (la última etapa es opcional), las cuales se describen a continuación en detalle. Al final de este apartado se incluye un gráfico que resume todo el proceso. Toda la lógica de detección y seguimiento se realiza dentro de la función de OpenCV onCameraFrame, la cual es llamada cada vez que se recibe un frame de la cámara. Esta función aparece como un método perteneciente a la clase que representa a cada activity.

- **Etapa 0**: comprobación de la orientación del dispositivo y limitación de la zona a procesar

En esta primera etapa el dispositivo comprueba sus ángulos de inclinación pitch y roll, los cuales se calculan en la biblioteca de estimación de distancias a partir del vector de gravedad. A medida que estos ángulos se desvían de cero, el error en el cálculo de distancias aumenta. Además, para aumentar el campo de visión de la aplicación, es necesario que el dispositivo se coloque en posición horizontal, con el lado más largo paralelo al plano del suelo. Por ello, cuando cualquiera de estos dos ángulos sobrepasa los 45º en sentido positivo o negativo, la aplicación deja de funcionar, no se detectan ni se siguen más objetos. La aplicación muestra un marco rojo alrededor de la pantalla para informar al usuario de este hecho. Para este proyecto el ángulo de inclinación pitch es el ángulo de giro sobre el eje y, mientras que el ángulo roll es el ángulo de inclinación sobre el eje z.

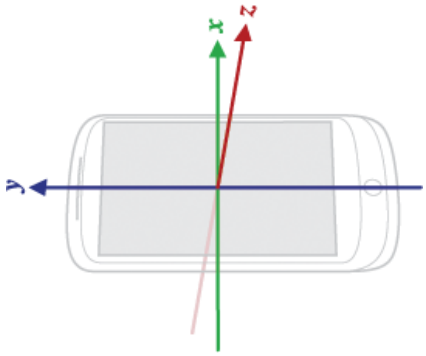


Figura 43: Sistema de coordenadas utilizado por los sensores [22]

La biblioteca de estimación de distancias calcula la distancia a un objeto a partir del punto central del borde inferior del rectángulo que contiene a dicho objeto, es decir, a partir del punto en el plano del suelo sobre el que se apoya el objeto. Este punto varía si el dispositivo se inclina sobre el eje y o sobre el eje z, pero no si la inclinación se produce sobre el eje x. La inclinación sobre el eje y (pitch) se tiene en cuenta en los cálculos de estimación de la distancia a través del método elaborado por Arturo de la Escalera. Debido al modo en el que está previsto que funcione el dispositivo (en posición horizontal, sobre el salpicadero o pegado a la luna de un automóvil), no se espera que existan inclinaciones sobre el eje z, por lo que simplemente se controla que sobrepase ciertos límites. Por último, el ángulo de inclinación sobre el eje x (yaw), no se tiene en cuenta al no influir en los cálculos de la estimación de distancias.

Posteriormente, si la inclinación del dispositivo es correcta, se procede a comprobar si existe un margen superior sobre el que no se deba realizar ninguna detección. Este margen existe porque es muy poco probable encontrar peatones o coches en la parte más alta de la pantalla, además de que este procedimiento reduce la cantidad de datos a procesar y aumenta el rendimiento de la aplicación. El margen se puede seleccionar manualmente o incluso se puede eliminar desde el menú de opciones de la aplicación. A partir de este margen se recorta la imagen original devuelta por la cámara y se convierte a escala de grises para ser procesada por los algoritmos de detección de coches o peatones.

- **Etapa 1**: detección de objetos mediante Haar Cascade u HOG

En esta etapa se comprueba que detector hay que usar y se llama a la función detectMultiScale de las clases CascadeClasifier u HOGDescriptor, según corresponda. Los parámetros que necesitan y sus valores se pueden consultar en el apartado 4.4. Solo es necesario indicar que los valores escogidos para la función detectMultiScale de la clase HOGDescriptor son los recomendados por N. Dalal and B. Triggs [18]. En el caso de utilizar el detector HOG se podrían utilizar imágenes a color, lo que mejoraría muy levemente los resultados a costa de incrementar el tiempo de procesamiento. Por este motivo se utilizan imágenes en escala de grises.

- **Etapa 2**: actualización de las matrices F y Q

Como ya se indicó en apartados anteriores, F es la matriz de transición de estados y Q es la matriz de covarianzas del error del modelo en el filtro de Kalman. Estas dos matrices tienen términos dependientes del tiempo, por lo que el programa calcula primero el tiempo de ciclo, es decir, el tiempo transcurrido entre dos frames consecutivos. Este tiempo también se puede ver como el tiempo que tarda la aplicación en procesar un frame. Con este dato se llama a la función actualizarFyQ, la cual calcula los valores de estas matrices para el frame actual. Como ya se ha indicado anteriormente, se ha utilizado un modelo de velocidad constante para modelar el movimiento de los objetos en seguimiento. A continuación se representan las ecuaciones y las matrices utilizadas, extraídas de [38]:

Modelo del sistema para aplicaciones basadas en coordenadas de la imagen

Modelo CWNA (Continuous White Noise Acceleration)

$$x_k = F_k x_{k-1} + w_k$$

$$\begin{pmatrix} U \\ \dot{U} \\ V \\ \dot{V} \end{pmatrix}_K = \begin{pmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} U \\ \dot{U} \\ V \\ \dot{V} \end{pmatrix}_{K-1} + w_k \rightarrow w_k \sim N\left(0, Q_k\right)$$

$$Q_k = \begin{pmatrix} \frac{1}{3}dt^3 & \frac{1}{2}dt^2 & 0 & 0 \\ \frac{1}{2}dt^2 & dt & 0 & 0 \\ 0 & 0 & \frac{1}{3}dt^3 & \frac{1}{2}dt^2 \\ 0 & 0 & \frac{1}{2}dt^2 & dt \end{pmatrix} \times \tilde{q}$$

Modelo del sistema para aplicaciones basadas en coordenadas reales

Modelo DWNA (Discrete White Noise Acceleration)

$$x_k = F_k x_{k-1} + G_k w_k$$

$$\begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{pmatrix}_K = \begin{pmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{pmatrix}_{K-1} + G_k w_k \rightarrow G_k w_k \sim N\left(0, Q_k\right)$$

$$G_{x,k} = G_{y,k} = \begin{pmatrix} \frac{1}{4}dt^4 & \frac{1}{2}dt^3 \\ \frac{1}{2}dt^3 & dt^2 \end{pmatrix} \rightarrow Q_k = G_k \sigma^2 G_k{}^T \rightarrow Q_k = \begin{pmatrix} \frac{1}{4}dt^4 & \frac{1}{2}dt^3 & 0 & 0 \\ \frac{1}{2}dt^3 & dt^2 & 0 & 0 \\ 0 & 0 & \frac{1}{4}dt^4 & \frac{1}{2}dt^3 \\ 0 & 0 & \frac{1}{2}dt^3 & dt^2 \end{pmatrix} \times \sigma^2$$

Figura 44: Modelo del sistema para el filtro de Kalman

Como se puede observar, para las aplicaciones basadas en coordenadas de la imagen se ha utilizado el modelo CWNA. Este modelo supone que la velocidad puede presentar variaciones en forma de aceleraciones, las cuales se modelan como ruido blanco de media cero a través del vector w_k. El único parámetro cuyo valor hay que calcular es el de $\tilde{q}$, que representa la intensidad del ruido del proceso, la cual se supone constante. La dimensión física de $\tilde{q}$ es [pixeles]^2/[segundo]^3. Un valor de 1E5 para peatones y de 1.5E5 para coches ofrece buenos resultados.

Por otra parte, para las aplicaciones basadas en coordenadas reales se ha utilizado el modelo DWNA. En este modelo el ruido del sistema está dividido en G_k y en w_k, siendo G_k un vector de ganancias del ruido y w_k una secuencia blanca de aceleraciones de media cero. Este modelo supone que existe una aceleración constante para cada periodo y que estas aceleraciones no presentan correlación periodo tras periodo. El motivo principal de su elección es que σ tiene dimensiones de aceleración [metros]/[segundo]^2, lo que facilita su elección. Se ha elegido un valor de 11 m/s^2 para peatones y de 18 m/s^2 para coches.

- **Etapa 3:** Predicción de la posición a partir del filtro de Kalman

Para cada track contenido en la lista de tracks en seguimiento se actualiza el valor de las matrices F y Q, según lo calculado en el apartado anterior, y se calcula la predicción del estado del track en el frame actual (valores de posición y velocidad). Estos cálculos los realiza la biblioteca JKalman llamando a la función Predict.

Si la aplicación funciona en coordenadas de la imagen, la predicción está representada por el rectángulo que envuelve al objeto. El estado de sistema (peatón o coche) está representado por la posición del pixel central de dicho rectángulo, que es lo que devuelve el filtro, junto con las velocidades asociadas para cada eje. El nuevo recuadro envolvente, la predicción, se calcula a partir de la posición predicha por el filtro y del último valor de ancho y alto conocido para para cada track.

Por el contrario, si la aplicación funciona en coordenadas reales, el estado predicho está formado por la estimación de las coordenadas reales x e y, así como de la estimación de las velocidades en dichos ejes. Durante el desarrollo de este proyecto se ha diseñado una función en la biblioteca de estimación de distancias que calcula la posición del punto de interés a partir de las coordenadas X e Y reales, para cada uno de los tres métodos que se pueden utilizar. Es decir, realiza la función inversa de las funciones originalmente presentes en la biblioteca. Con este punto, y conociendo el último valor de anchura y altura del objeto, se genera el rectángulo que representa la nueva posición del track.

En este apartado también se calcula el gat, o lo que es lo mismo, la máxima distancia entre una predicción y una detección para que estas puedan ser asociadas. La forma más correcta de calcular este dato es multiplicar la desviación típica de la posición por un cierto valor constante elegido empíricamente. La desviación típica de las posiciones y las velocidades se puede calcular como la raíz cuadrada de los elementos de la diagonal principal de la matriz P del filtro (matriz de covarianzas del error de estado). El gat para las coordenadas horizontal y vertical se calcula como $2 \cdot desv(coordenada)$.

- **Etapa 4:** filtrado de las nuevas detecciones y cálculo de la distancia a estas

Anteriormente ya se comentó que el punto de interés de un objeto está definido como el punto medio del borde inferior del rectángulo que envuelve al objeto. Es decir, es el punto de contacto entre el objeto y el plano del suelo. A partir de este punto se calcula la distancia a la que se encuentra el objeto, por lo que debe estar calculado de la forma más precisa posible. El recuadro que rodea a un objeto es la salida de los algoritmos de detección, por tanto no

podemos influir directamente en ellos. Por suerte, cuando se utiliza el detector Haar Cascade, este recuadro suele definir correctamente el espacio ocupado por el objeto. Sin embargo, cuando se detectan peatones con HOG, este recuadro es siempre más grande que el peatón detectado por la forma en la que está implementado el algoritmo. Por ello es necesario adaptar primero el tamaño de este recuadro antes de continuar. Reduciendo la altura un 15 % y la anchura del recuadro un 30 % se consigue que se adapte mejor al tamaño real del peatón, acercando el punto de interés a su lugar correcto.

Los algoritmos de detección en ocasiones detectan dos veces a la misma persona, encuadrándola en dos o más rectángulos distintos superpuestos. Para evitar que esto afecte a fases posteriores del proceso, este problema es eliminado llamando a la función filtrarDetecciones, incluida en la clase Deteccion. Para ello se pasa a la función la lista de nuevas detecciones encontradas en el frame actual. La función coge cada detección y comprueba si está superpuesta con otra; si es el caso una de ellas ha de ser eliminada. Si se utiliza HOG es proceso es muy sencillo, pues el propio algoritmo devuelve una lista con los pesos de cada detección. Estos pesos no son más que el grado de seguridad que el algoritmo tiene de haber encontrado el objeto buscado. La detección con mayor peso sobrevive. En el caso de utilizar Haar Cascade es más complicado, puesto que no se informa de manera alguna sobre el grado de seguridad de la detección. En este punto se comparan las dos detecciones solapadas con las predicciones del filtro calculadas en la etapa anterior. Una vez comparadas las dos detecciones con todas las predicciones, si el grado de solapamiento de la detección con mayor solapamiento a cualquiera de las predicciones es mayor que 0.7, esta se dará por válida y la otra se eliminará. Si esto no se da, se realiza la media entre las dos detecciones.

Por último, en esta etapa se calcula la distancia a cada detección en coordenadas reales, pues hasta ahora la aplicación solo conoce para cada detección el rectángulo que la envuelve. Esta tarea la realiza la función crearLista de la clase Deteccion. En ella se calcula el punto de interés para cada detección y la distancia a la que se encuentra a partir de la biblioteca de estimación de distancias. Cada detección se convierte en un objeto de la clase Deteccion y se incluyen todas en una lista o vector, que es el resultado final de la función crearLista.

- **Etapa 5:** asignación de las nuevas detecciones con algún track en seguimiento

En esta etapa básicamente se aplica el Algoritmo Húngaro. Lo primero que se comprueba es el número de detecciones y tracks que existen actualmente. Como es lógico, estos dos valores no tienen por qué coincidir, puesto que pueden aparecer detecciones nuevas (más detecciones que tracks) o puede haber tracks ocluidos o simplemente invisibles que no puedan asociarse a ninguna detección (más tracks que detecciones). El método Húngaro sin embargo asocia en una relación uno a uno filas con columnas, por lo que la matriz de costes debe ser obligatoriamente cuadrada. En este caso la dimensión de la matriz se escoge como el máximo entre el número de tracks y el de detecciones, por lo que habrá filas o columnas fantasma representando a tracks o detecciones ficticios.

Una vez escogida la dimensión de la matriz de costes, esta se genera inicializando todos los valores de las celdas al valor lo más alto posible, en este caso llamado *impossible_assigment* y cuyo valor es la constante Integer.MAX_VALUE. El siguiente paso es recorrer las listas de tracks

y de detecciones e ir calculando la distancia entre todos ellos. Es decir, el valor de la celda [0,0] de la matriz será la distancia entre el track en la posición 0 de la lista de tracks y la detección en la posición 0 de la lista de detecciones. Si al calcular la distancia entre un track y una detección se supera para alguna coordenada el gat asociado a esta, se considera que la asociación no es posible, de tal manera que la celda correspondiente queda con un valor de *impossible_assigment*. Esto mismo ocurre con aquellas celdas que están asociadas a filas o columnas ficticias. Un valor tan alto de coste implica que estos elementos ficticios no influirán en la asociación entre tracks y detecciones reales.

Después de crear la matriz de costes, esta se envía al Algoritmo Húngaro, el cual resuelve el problema dando como salida un vector en el que cada posición representa el índice de un track y cada valor de dicha posición representa el índice de la detección asociada.

$$\begin{bmatrix} 0 & 4 & 3 & 1 \end{bmatrix}$$

En el ejemplo anterior podemos observar, si nos fijamos en el primer número, que el algoritmo ha asociado el track con índice 0 con la detección con índice 0. El segundo valor nos indica que el track 1 ha sido asociado con la detección 4 y así con el resto.

Una vez que el algoritmo ha entregado su solución, los pasos a seguir para discernir entre asociaciones reales, tracks invisibles y detecciones sin asignar son los siguientes:

- Para cada asociación que realiza el algoritmo se comprueba la distancia entre cada pareja de track y detección en la matriz de costes anteriormente creada. Si esta distancia es igual a *impossible_assigment* se considera que la asociación no es válida, en cualquier otro caso se considera válida y se añade a una lista de asociaciones. Las asociaciones no válidas se producen por el hecho de que el algoritmo está diseñado para asociar filas con columnas en una relación de uno a uno de forma tal que no pueden existir ni filas ni columnas sin asociar. Cuando acurre esto la lógica a seguir es la siguiente:
    - Si ocurre que el índice del track es mayor o igual que el número de tracks reales, implica que este track realmente no existe, por lo que la detección se añade a la lista de nuevas detecciones, las cuales podrán convertirse después en nuevos tracks.
    - Si ocurre al contrario, es decir, que el índice de la detección es mayor o igual que el número máximo de detecciones, significa que la detección no es real, por tanto el track se considera como un track invisible.
    - Por último, si no se cumple ninguna de las anteriores dos condiciones, significa que la asociación no válida se ha realizado entre un track real y una detección real. El no poder asociarlos implica que el track es en realidad un track invisible a la vez que la detección es nueva, y por tanto no puede asociarse a ningún track.

El algoritmo usado es una adaptación del algoritmo creado por Myriam Abramson [42], el cual es a su vez una adaptación en Java del algoritmo creado por Bob Pilgrim [43], implementado en C#. El código estaba diseñado para ser utilizado en una aplicación de escritorio, por lo que se tuvieron que realizar varias modificaciones para poder adaptarlo a su uso en este proyecto.

- **Etapa 6:** Corrección de errores y actualización del estado de los tracks

El primer paso dentro de esta etapa consiste en comprobar la existencia de oclusiones. Es importante distinguir entre tracks invisibles y tracks ocluidos, pues estos últimos se mantienen más tiempo en seguimiento, a la espera de que se termine la oclusión. Esta aplicación considera como oclusión cualquier track que, siendo invisible, está solapado o superpuesto a cualquier otro track que se encuentre asociado a cualquier detección.

Seguidamente se llama a una función llamada *corregirErrores*, la cual, como su propio nombre indica, intenta resolver el que es quizá uno de los mayores problemas que tenía la aplicación en sus comienzos. Los detectores utilizados no son perfectos, por lo que en ocasiones sucede que detectan un mismo objeto en dos frames consecutivos otorgándole dos bounding boxes distintos. Esto ocurre con mucha más frecuencia en el caso del detector Haar Cascade. Por este motivo ocurre que un objeto, un peatón por ejemplo, está siendo seguido por la aplicación cuando de repente en un frame el recuadro generado por el detector cambia de tamaño súbitamente. Cuando esto ocurre la asociación ya no es posible, la aplicación marca el antiguo track como invisible y crea uno nuevo a partir de la nueva detección. Tras varios frames el recuadro generado alrededor del peatón vuelve a cambiar su tamaño o posición, por lo que el último track creado se etiqueta como invisible y el primero vuelve a ser asignado a una detección. El resultado de este proceso es que un mismo objeto tiene asociados dos o más tracks que van alternando entre estados de visibilidad e invisibilidad.

Para solucionar el problema anterior la aplicación comprueba el solapamiento existente entre los recuadros de las nuevas detecciones y aquellos pertenecientes a tracks marcados como invisibles, pero no ocluidos. Si este valor calculado es grande (mayor del 75%), se mantiene el track actual y se modifica ligeramente el tamaño y la posición de la nueva detección en función de los valores de dicho track. En resumen, tenemos un track invisible que tiene superpuesta una detección lo suficientemente distinta como para que el Algoritmo Húngaro no los haya asociado de forma automática (la detección estaba fuera del gat del track). Sin embargo lo más probable es que se trate de un error del detector. Por ello se genera una nueva detección mediante una media ponderada entre el track invisible y la propia detección en una relación 80 – 20. Más tarde el estado del track será actualizado a partir de esta detección modificada. Con esto también se consigue que si el cambio en tamaño de la detección es real por cualquier motivo, el track en vez de desaparecer se vaya ajustando a su nuevo tamaño y/o posición.

A continuación la aplicación toma la lista de asignados y la hace efectiva. Con ello la aplicación es capaz de actualizar el estado de los tracks asignados, mejorando la precisión del tracking y la estimación de la posición de estos. Como ya se ha comentado en el anterior párrafo, los recuadros envolventes a los objetos pueden variar bastante de un frame a otro si se utiliza como detector Haar Cascade. Para evitar este problema lo que se hace es realizar una media entre las tres últimas detecciones y tomar esta como la detección a partir de la cual se extrae la posición del objeto, posición que se utiliza para actualizar el estado del track.

En este último apartado cabe destacar que, cuando se utiliza la aplicación basada en coordenadas reales, también se actualiza junto con el estado la matriz R (matriz de covarianzas del error de medida). Esto es necesario porque el algoritmo que estima la distancia a los objetos presenta más error cuanto mayor es la distancia. Esto ocurre porque dicha distancia se calcula a partir de la diferencia en píxeles entre el centro de la imagen y el punto de interés, medida en el eje V. De esta forma, cuanto mayor es la distancia a la que se encuentra el objeto, mayor es

la variación de esta por cada pixel de diferencia entre el centro de la imagen y el punto de interés. La aplicación calcula la desviación típica del error en cada coordenada como la distancia en dicha coordenada dividida por ocho. Si se está utilizando HOG la covarianza se disminuye en un 10 % respecto al valor anterior.

$$\begin{pmatrix} \left(\frac{X}{8}\right)^2 & 0 \\ 0 & \left(\frac{Y}{8}\right)^2 \end{pmatrix}, \begin{pmatrix} 0.9\cdot\left(\frac{X}{8}\right)^2 & 0 \\ 0 & 0.9\cdot\left(\frac{Y}{8}\right)^2 \end{pmatrix}$$

Figura 45: matriz R del filtro de Kalman. Coordenadas reales.

Para acabar esta etapa la aplicación actualiza el estado de los tracks que se quedan sin asignar. El objetivo de esta parte es básicamente eliminar aquellos tracks invisibles que ya no sean considerados válidos. Esto ocurre cuando se da al menos una de estas tres condiciones:

- El track ha estado invisible durante 5 frames consecutivos.
- El track ha estado ocluido durante más de 5 segundos también consecutivos.
- Durante los 8 primeros frames el track ha permanecido invisible en más del 45 % de las ocasiones.

- **Etapa 7:** creación de nuevos tracks

Una vez actualizados todos los tracks existentes, se generan nuevos posibles tracks a partir de todas las detecciones que han quedado sin asignar. Si en el siguiente frame estos nuevos tracks pueden ser asignados a una detección, se constituirán como verdaderos tracks y serán mostrados por la aplicación. Si esto no sucede, estos serán eliminados de forma automática al llegar a la etapa 6. Este proceso disminuye el número de falsos positivos que presenta la aplicación.

Cuando se crean nuevos tracks se tienen en cuenta las siguientes consideraciones:

- La matriz de transición de estados F se inicializa como una matriz unidad de orden 4.
- El vector de estado se inicializa a partir de la posición de la detección y la velocidad se considera cero.
- Cuando un objeto es detectado y seguido en varios frames consecutivos, la matriz de covarianzas del error de estado P tiende a tomar un valor constante. La matriz de covarianzas del error de estado inicial P_0 se inicializa a este valor para aumentar la eficiencia inicial del filtro [38]. Otra opción sería iniciar la diagonal principal de esta última matriz con un valor alto, dejando el resto de celdas con valor cero. Con esto se consigue indicar de alguna forma al filtro que el error del estado es alto, por tanto prevalecerá la información de las primeras detecciones sobre la información actualmente contenida en el filtro [39].
- La matriz de medidas H se inicializa al siguiente valor:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- La matriz Q se inicializa de la forma indicada en la etapa 2 (el constructor de la clase Tracking recibe como parámetro el tiempo de ciclo de la aplicación).
- En el caso de las aplicaciones basadas en coordenadas reales la matriz R se inicializa según lo indicado en la etapa anterior. En el caso de utilizar las aplicaciones basadas en coordenadas de la imagen, el valor de esta matriz es el siguiente:

$$\begin{pmatrix} 0.85 \cdot W + 30 & 0 \\ 0 & 0.95 * \mathrm{H} - 120 \end{pmatrix}, \begin{pmatrix} 0.8 * W & 0 \\ 0 & 0.9 * H \end{pmatrix}$$

Figura 46: matriz R del filtro de Kalman. Coordenadas de la imagen.

La matriz de la izquierda corresponde al valor de R cuando se utiliza el detector Haar Cascade, mientras que la matriz de la derecha se utiliza junto con HOG. W es el ancho en píxeles de la imagen y H se corresponde con la altura. Para calcular estas matrices se realizó una prueba en la que el objeto detectado se mantenía estático. En dicha prueba el dispositivo móvil estuvo también varios minutos estático recogiendo la información sobre la posición del objeto. Como ninguno de los dos se mueve, todas aquellas variaciones en la posición del bounding box están causadas por el error de detección del detector, por lo que con estos datos se puede calcular el error de medida del mismo.

En todos los casos la matriz R es una matriz diagonal, lo que implica que se considera que la covarianza entre distintos ejes es cero. Esta simplificación se asemeja bastante a la realidad, pero el motivo principal de su uso es que disminuye la complejidad de los cálculos que debe realizar el filtro de Kalman, aumentando el rendimiento de la aplicación [38]. Ello es debido a que se simplifica el cálculo de matrices inversas que realiza el filtro internamente.

- **Etapa 8:** representación de los resultados en pantalla

En este apartado simplemente se muestran en pantalla todos aquellos tracks que en el frame actual estén visibles. Cada track tiene asociado desde su creación un color distinto para representar sus recuadros envolventes. Además también se muestra un cerco verde alrededor de la pantalla que indica la parte que se procesa, que no tiene por qué ser toda la imagen captada. Para más detalle de esto último se puede consultar el apartado 0.

- **Etapa 9:** registro de datos e imágenes (opcional)

Esta etapa es opcional, pues su función es la de registrar las imágenes y los datos de todos los objetos en seguimiento. Igualando el atributo *debugMode* presente en las cuatro clases principales a false se desactiva esta característica, que fundamentalmente es utilizada para realizar pruebas y mejoras en la aplicación.

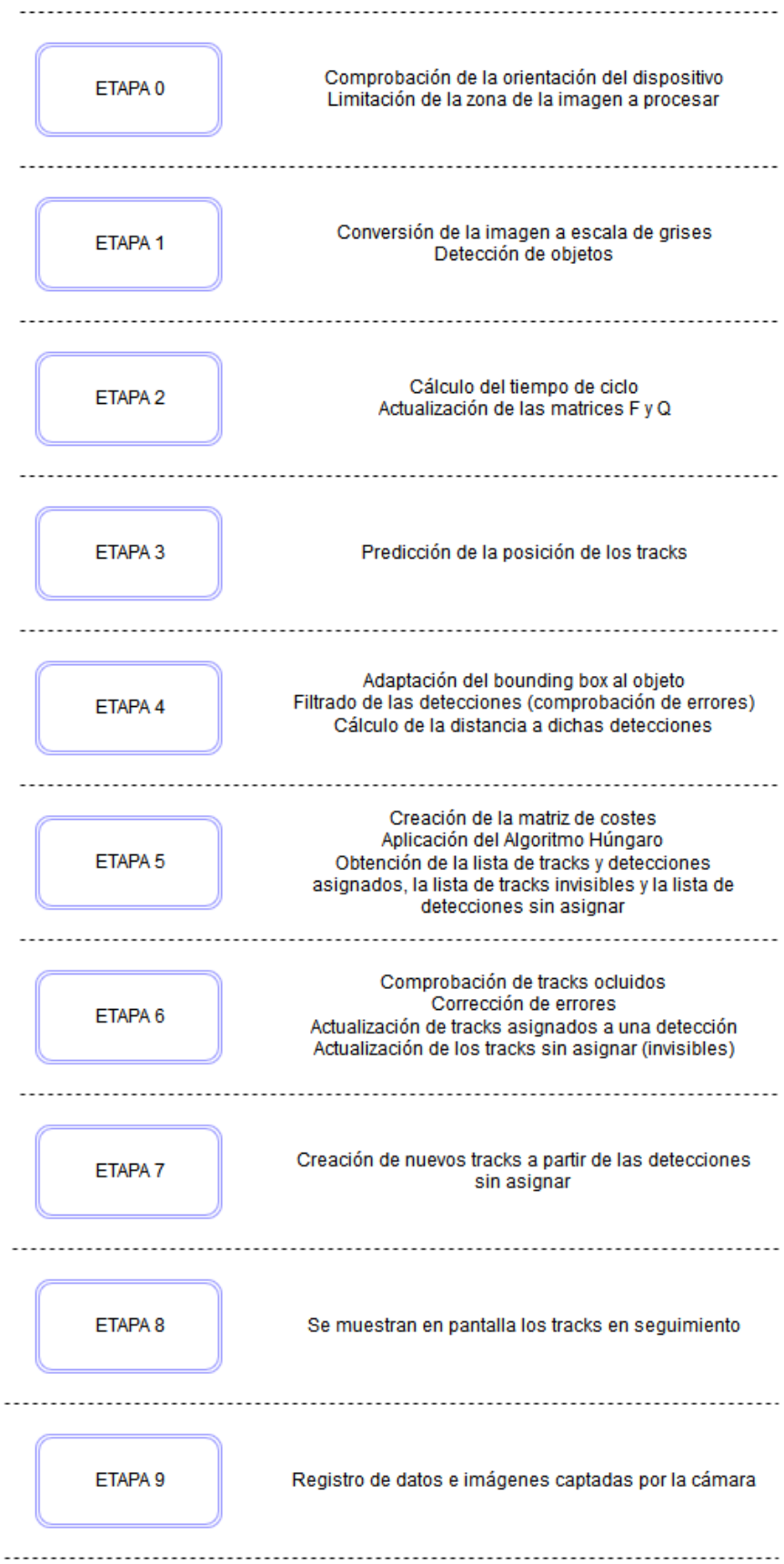


Figura 47: Resumen de las etapas de la aplicación

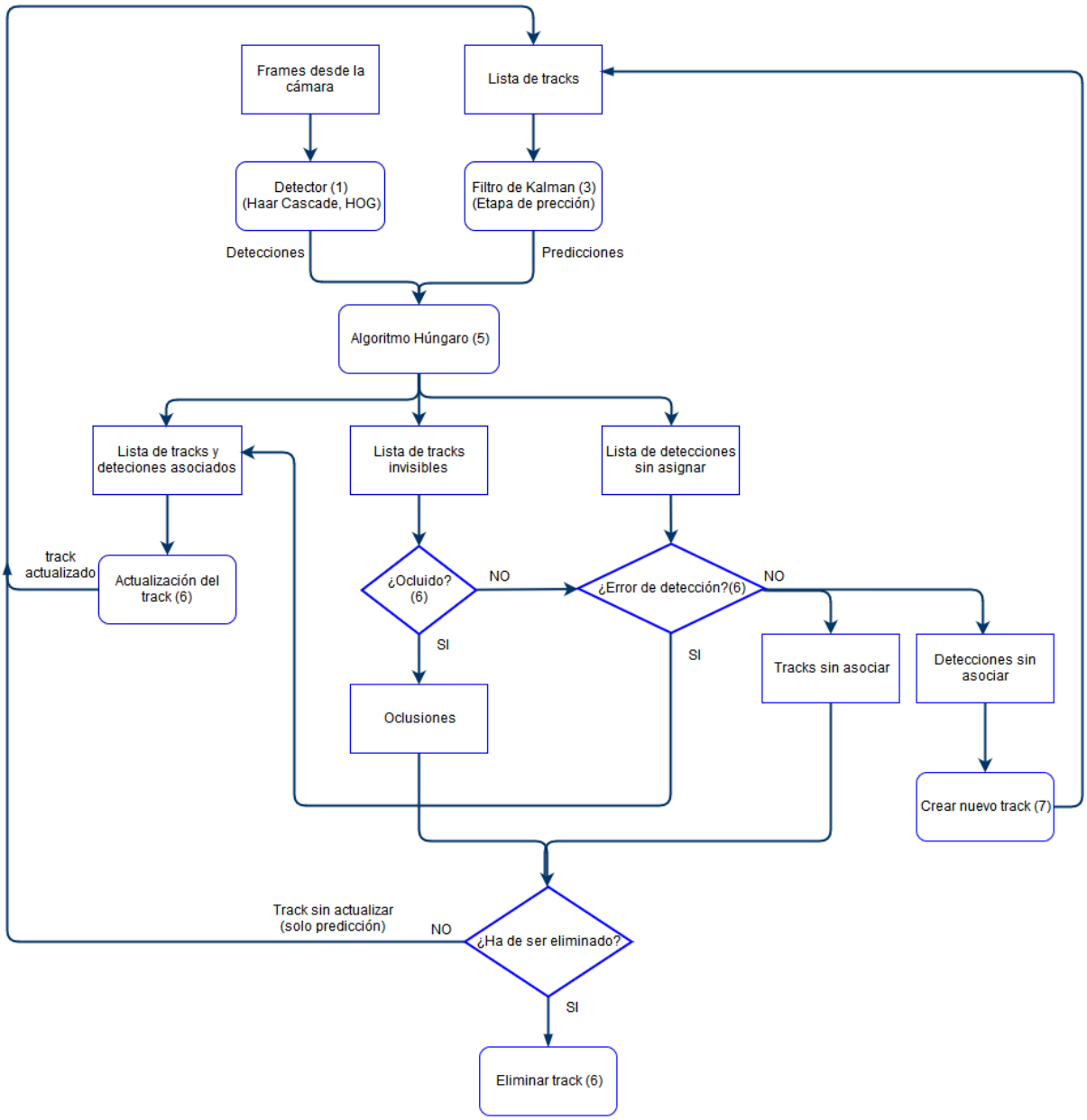


Figura 48: ciclo de vida de un track

En la imagen superior los números que aparecen entre paréntesis corresponden a la etapa en la cual se realiza la acción a la que se refieren.

5. RESULTADOS

En este apartado se recoge el resultado de las distintas pruebas realizadas a la aplicación una vez terminada. Estas pruebas tienen el objetivo de determinar el funcionamiento y la fiabilidad de la aplicación desarrollada, así como de determinar cuál es la configuración que mejor resultados ofrece. Las distintas configuraciones que permite la aplicación son cuatro: detección en coordenadas de la imagen y coordenadas reales utilizando el detector HOG y la detección en coordenadas de la imagen y coordenadas reales utilizando el detector Haar Cascade.

En primer lugar se analizarán los distintos factores ajenos a la propia aplicación que puedan afectar al rendimiento y la eficacia de la misma. Seguidamente se analizarán aquellos factores internos a la aplicación que ejercen un mayor efecto sobre los resultados obtenidos. Finalmente se ofrece una serie de gráficas y tablas donde se recogen los resultados de las pruebas efectuadas.

5.1. Factores limitantes ajenos a la aplicación

La eficacia que presenta la aplicación para realizar las labores de detección y seguimiento de coches y peatones está, o puede estar, limitada por una serie de condiciones ajenas al propio dispositivo o aplicación. A continuación se ofrece una lista de los factores más relevantes:

- Condiciones meteorológicas: en un día lluvioso, muy nublado, con presencia de niebla o, de forma más general, con visibilidad reducida, la detección de objetos se ve afectada negativamente. Como es lógico de noche, al atardecer o anochecer la aplicación es incapaz de realizar detecciones debido a la falta de luz. En estas condiciones una cámara de infrarrojos sería el dispositivo más apropiado para realizar dichas detecciones.
- Sombras y reflejos: causantes también de fallos en la detección.
- Oclusiones: cuando el objeto a detectar está parcialmente ocluido por otros objetos, es probable que el detector no sea capaz de identificar a dicho objeto, pues esta oclusión impide al detector encontrar aquellas características que busca.
- Forma de la vía: la biblioteca de estimación de distancias está basada en un modelo de suelo plano, por lo que si la vía sobre la que se circula presenta grandes pendientes respecto al vehículo sobre el cual se monta el dispositivo móvil, esta estimación tendrá un error. Mayor será el error cuanto mayor sea la inclinación del terreno.
- Paisaje de fondo: si el fondo de la escena captada por la cámara presenta una gran cantidad de irregularidades o ruido, la capacidad de detección se ve reducida. Lo mismo ocurre si el fondo presenta un color muy similar al objeto a detectar.

Estos factores pueden llegar a afectar seriamente al funcionamiento de la aplicación, pero desgraciadamente no son dependientes del usuario, ni del desarrollador, ni del propio dispositivo, por lo que son realmente complicados de manejar.

5.2. Factores limitantes propios del dispositivo o de la aplicación

Las limitaciones procedentes del dispositivo son básicamente limitaciones tipo hardware. Las más importantes son el procesador, la memoria y la cámara. Si el dispositivo no cuenta con unos requerimientos de capacidad de cálculo y memoria adecuados, la aplicación no será capaz de funcionar en tiempo real, con lo que no podría utilizarse. Igualmente, si la cámara no cuenta con la resolución suficiente, tanto la detección de objetos como la estimación de distancias presentarán errores.

A partir de las pruebas efectuadas en distintos dispositivos se puede concluir que la aplicación se puede ejecutar en dispositivos móviles que tengan al menos un procesador de un 1 GHz, una memoria de 1 GB y una cámara de unos 5 megapíxeles.

Los dispositivos comerciales que realizan una función similar a la aquí desarrollada no presentan este tipo de problemas debido a que utilizan un hardware dedicado. Estos dispositivos se comercializan en un paquete en el que se incluye una cámara específica y un procesador y una memoria dedicados, diseñados especialmente para la tarea a realizar. Gracias a esto el rendimiento es siempre el mismo. Como es evidente, en el caso de la aplicación desarrollada, no se puede garantizar que el dispositivo sobre el que se vaya a ejecutar cumpla con los requisitos adecuados, afectando este hecho al rendimiento.

En cuanto a los factores limitantes propios de la aplicación se puede destacar la salida del detector utilizado. Si el recuadro que envuelve al objeto a detectar no se ajusta adecuadamente al propio objeto, la estimación de distancias tendrá un error, tal y como se explica en el apartado 4.8. De hecho esta es la principal fuente de error en dicha biblioteca. Se trata de un problema difícil de resolver, ya que la fiabilidad de la detección depende de los factores ajenos tratados en el apartado anterior. Otra fuente de error en la biblioteca de estimación de distancias viene dada por una excesiva inclinación del dispositivo. Dado que la aplicación está diseñada para ser utilizada en un dispositivo anclado a la luna delantera del coche, esto no debería ser un problema.

En cuanto al proceso de seguimiento, el principal problema que puede afectar al filtro de Kalman consiste en que el dispositivo no sea capaz de mover la aplicación con soltura. Cuanto esto ocurre, la matriz Q aumenta su valor, con lo que la predicción de la posición que realiza el filtro presenta mayor incertidumbre. En casos extremos esto puede ocasionar que el filtro no sea capaz de seguir a los objetos detectados.

Por último, en cuanto a los sistemas de detección, es importante indicar que el archivo de entrenamiento utilizado con el detector Haar Cascade está entrenado para la detección de personas que se encuentren de cara o de espaldas a la cámara. Esto no ocurre con el detector HOG. Otro punto a favor de este último detector es que se basa en calcular y operar con la orientación de los gradientes en la imagen (los bordes). Estos bordes están presentes aun cuando se reduce considerablemente la resolución de la imagen, lo que puede salvar en cierto modo el problema de que su rendimiento sea menor.

5.3. Pruebas realizadas

Para comprobar la fiabilidad del sistema de estimación de distancias y de seguimiento basado en el filtro de Kalman, se ha utilizado el esquema disponible más abajo. En él se puede observar que las pruebas realizadas consisten en registrar el movimiento de varias personas realizando una serie de trayectorias horizontales, verticales y diagonales de ida y vuelta.

En las trayectorias verticales (en el eje y) se conoce en todo momento el valor de la coordenada x, pues es fija. Con esta trayectoria podemos por tanto conocer la fiabilidad del sistema de seguimiento y estimación de distancias en el eje x. En las trayectorias horizontales (eje x) la coordenada fija es la y, por lo tanto esta es la coordenada sobre la que se trabaja. Con este esquema se puede obtener en todo momento una comparación entre las coordenadas reales y la posición estimada del objeto para distintas trayectorias en ambos ejes, obteniéndose así el error de seguimiento en cada instante. En los movimientos diagonales se han recorrido las trayectorias a velocidad constante, lo que, junto con los datos registrados por la aplicación, ha permitido realizar una comparación entre la distancia real y la estimada en este tipo de movimientos.

Para obtener el ratio de detección y el número de falsos positivos de la aplicación se ha activado el modo debug de la misma (apartado 4.10) el cual permite registrar en un fichero de texto los datos referentes a cada detección junto con el tiempo en que se produce dicha detección. Además permite grabar las imágenes que la aplicación ha procesado.

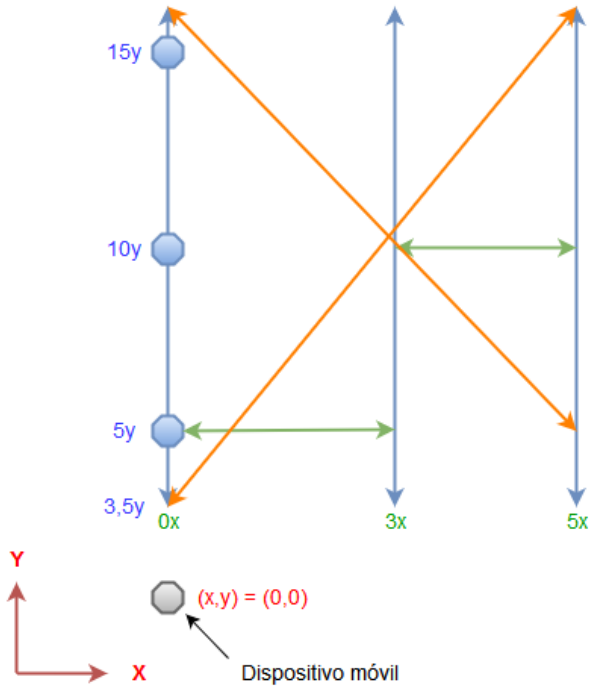


Figura 50: Esquema de las pruebas realizadas

En el esquema superior cada flecha representa una trayectoria. Todas las trayectorias comienzan al menos a 3.5 metros del dispositivo móvil en el eje 'y' para que desde un primer momento la persona sea detectada y tener un punto de inicio en el que comenzar a comparar

las coordenadas reales con las estimadas por la aplicación. El sistema de coordenadas de la imagen y de coordenadas reales se explica con más detalle en el apartado 4.3.

5.4. Resultados obtenidos

A continuación se presentan los resultados más significativos obtenidos tras realizar las pruebas descritas en el apartado anterior para distintas configuraciones de la aplicación. En primer lugar se mostrará una tabla en la que se resume el resultado de las pruebas realizadas con el fin de calcular la matriz de covarianzas del error de medida (o matriz R) para el filtro de Kalman. A continuación se presentan una serie de gráficas en las que se puede ver una comparación entre la trayectoria real y la trayectoria seguida por la aplicación, así como el error en cada instante. Para terminar se muestra una tabla que contiene datos sobre un mayor número de pruebas realizadas. Finalmente se presentan una serie de conclusiones extraídas a partir de los datos obtenidos en las pruebas.

Una de las matrices que hay que calcular para poder utilizar el filtro de Kalman es la matriz R o matriz de covarianzas del error de medida. Con esta matriz indicamos al filtro de Kalman el error que existe en la medida de la posición de los objetos detectados. El procedimiento para calcularla consiste en dejar a una persona parada delante de la cámara y calcular la variación de la posición obtenida a través del detector. Para ello hay que calcular la desviación típica y la covarianza de las coordenadas del peatón durante el tiempo que dura la prueba. En la siguiente tabla se muestran los resultados.

DISTANCIA (m)	RESOLUCIÓN	D. TÍPICA (X)	D. TÍPICA (Y)	COV(X)	COV(Y)
Haar Cascade, Coordenadas reales (X,Y), medidas en metros					
5	800 x 480	0.010	0.165	0.00010	0.027
8	800 x 480	0.0063	0.287	3.98E-05	0.082
10	800 x 480	0.3963	1.26	1.157	1.595
5	480 x 320	0.0046	0.128	2.15E-05	0.0165
8	480 x 320	0.01	0.288	0.0001	0.083
10	480 x 320	0.629	2.42	0.396	5.85
HOG, Coordenadas reales (X,Y), medidas en metros					
5	384 x 288	0.225	0.216	0.05	0.047
8	384 x 288	0.012	0.16	0.00014	0.0264
RESOLUCIÓN		**D. TÍPICA (U)**	**D. TÍPICA (V)**	**COV(U)**	**COV(V)**
Haar Cascade, Coordenadas de la imagen (U,V), medidas en píxeles					
800 x 480		6.23	4.15	37.56	17.32
480 x 320		1.08	3.54	1.17	12.53
384 x 288		1.118	1.76	1.25	3.11
HOG, Coordenadas de la imagen (U,V), medidas en píxeles					
384 x 288		2.93	0.707	8.57	0.5

Tabla 2: Datos para el cálculo de la matriz R

Al realizar distintas pruebas en las que se modifica tanto la resolución como la distancia, somos capaces de generar en el programa matrices R capaces de adaptarse a cambios en estos dos parámetros. En los resultados obtenidos se puede apreciar como la desviación típica y la varianza suelen ser mayores en la coordenada vertical que en la horizontal. Esto se debe a que la variación del recuadro envolvente al objeto es mayor casi siempre en esta coordenada.

Las siguientes gráficas muestran, para distintas configuraciones y trayectorias, una comparación entre la trayectoria real y la seguida por la aplicación. Además se indica el error de seguimiento que se produce en cada momento. El número de pruebas realizadas es mucho mayor, pero aquí solo se muestran algunas, por ser las más significativas y por cuestiones de espacio.

- Trayectoria x = 0, coordenadas de la imagen (U,V), detector Haar Cascade

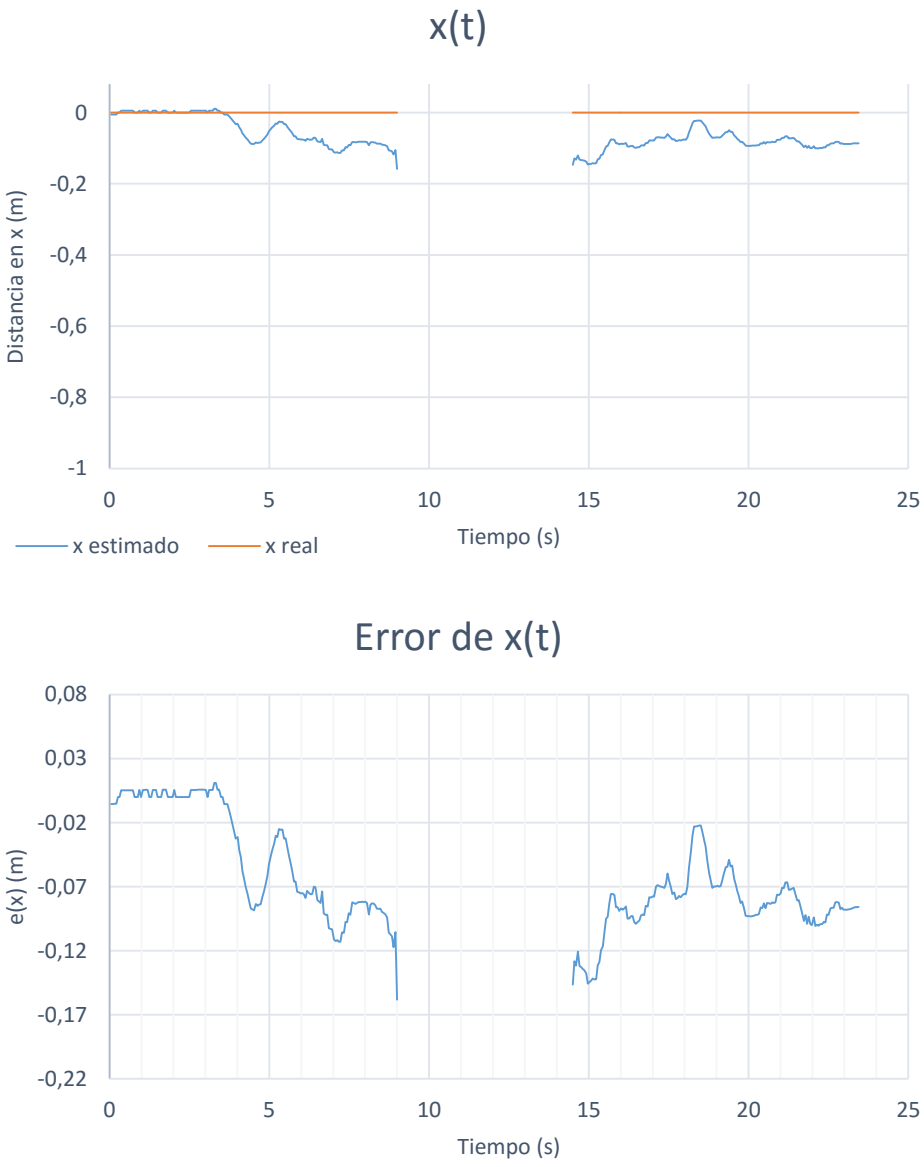


Figura 51, gráficas 1 y 2: Trayectoria x = 0 seguida en el sistema de coordenadas de la imagen y utilizando el detector Haar Cascade

El espacio intermedio vacío de medidas corresponde al tiempo en el que el peatón se aleja lo suficiente del dispositivo como para que no sea detectable por el mismo. Como se indica más adelante, la aplicación detecta correctamente a los peatones hasta una distancia de aproximadamente 12 metros. En las pruebas realizadas cuya trayectoria tiene la coordenada x fija (movimiento de alejamiento y acercamiento en el eje y) el sujeto de pruebas se alejaba del dispositivo hasta una distancia de unos 15 a 20 metros en su trayectoria de ida, y luego regresaba hacia la posición inicial. Por tanto, el espacio vacío de medidas corresponde al tiempo en el que el peatón se encuentra en el rango de 12 a 15 o 20 metros del dispositivo.

- Trayectoria x = 0, coordenadas reales (X,Y), detector HOG

Figura 52, gráficas 3 y 4: Trayectoria x = 0, seguida en coordenadas reales y utilizando el detector HOG

- Trayectoria x = 5, coordenadas de la imagen (U,V), detector CASCADE

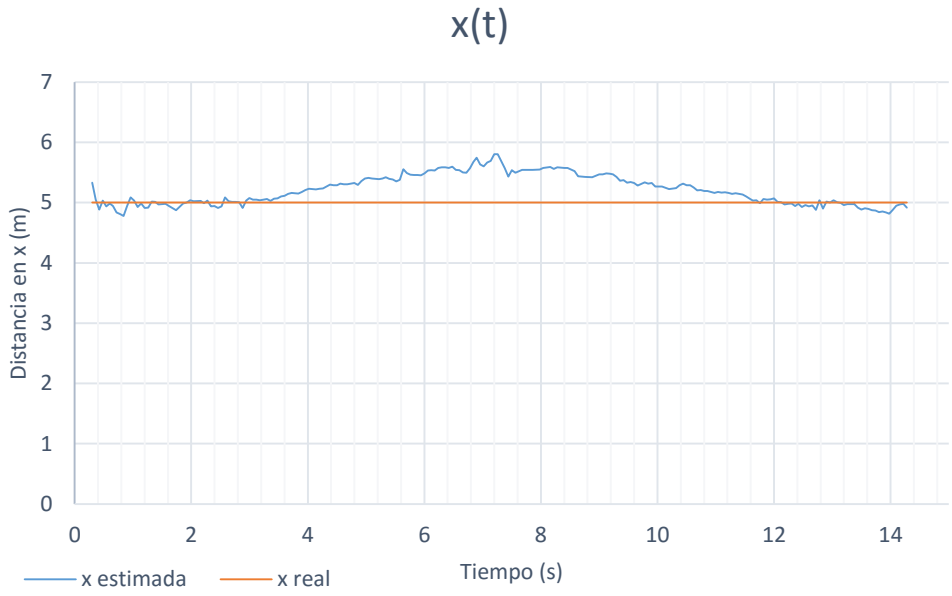
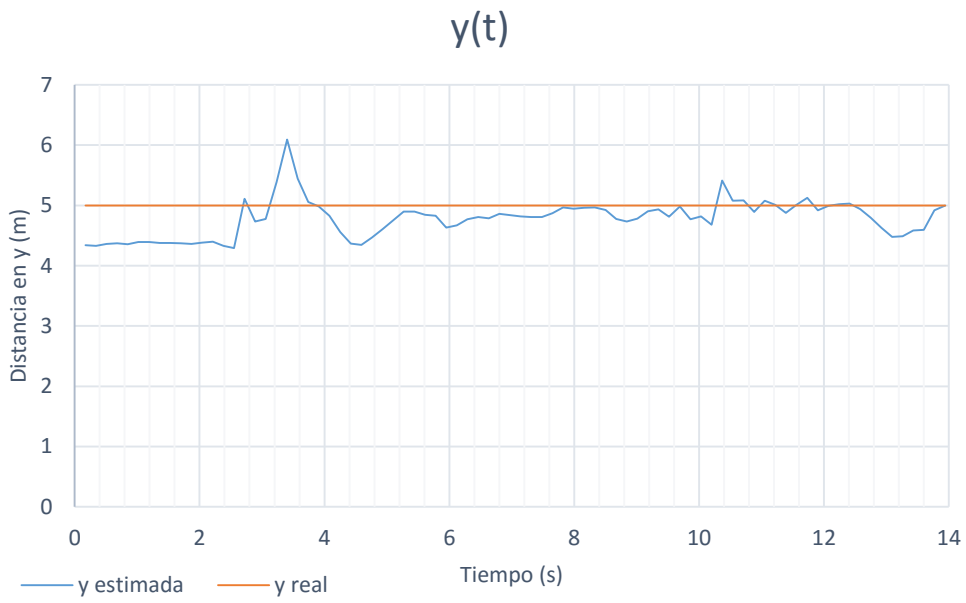


Figura 53, gráficas 5 y 6: Trayectoria x = 5, seguimiento realizado en coordenadas de la imagen, utilizando el detector Haar Cascade

- Trayectoria y = 5, coordenadas reales (X,Y), detector HOG



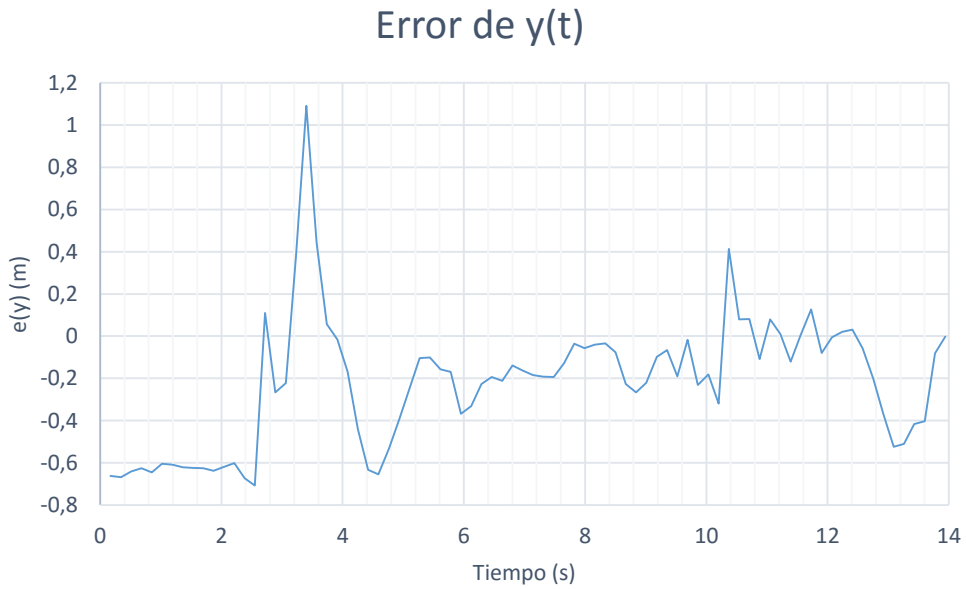
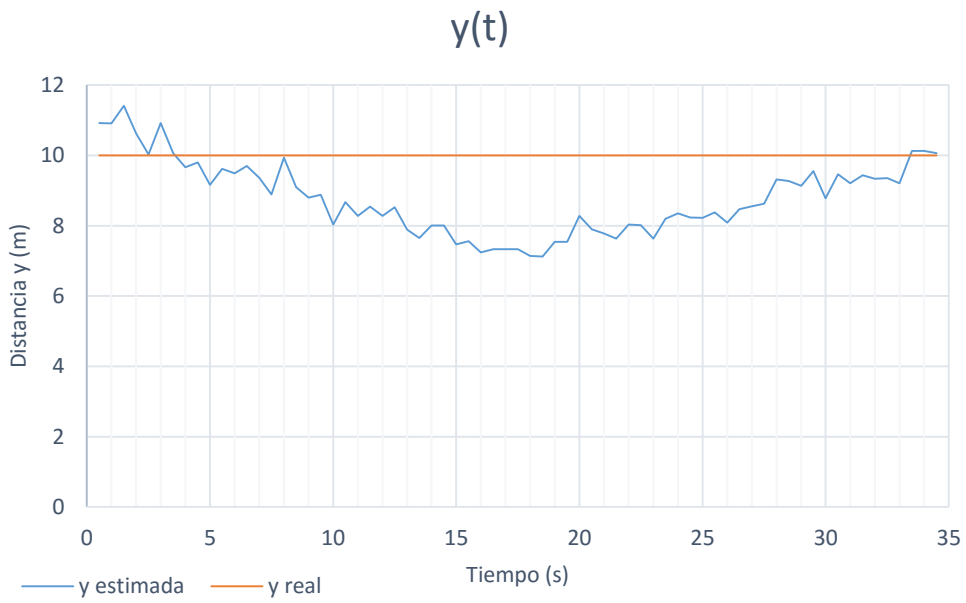


Figura 54, gráficas 7 y 8: Trayectoria y = 5, seguimiento en coordenadas reales, utilizando el detector HOG

- Trayectoria y = 10, coordenadas de la imagen (U,V), detector CASCADE



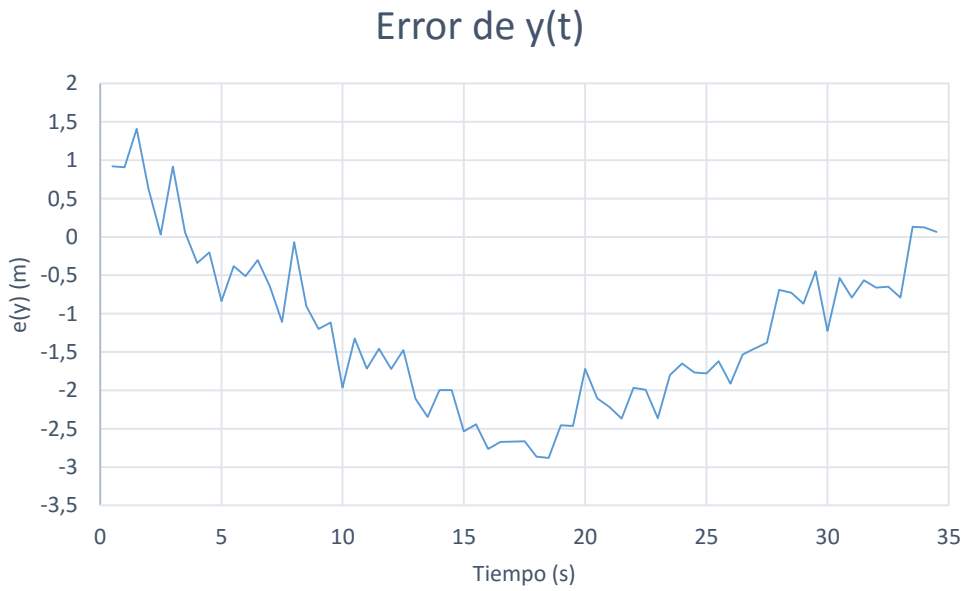


Figura 55, gráficas 9 y 10: Trayectoria y = 10, seguimiento en coordenadas de la imagen, utilizando el detector Haar Cascade

- Trayectoria diagonal, coordenadas reales (X,Y), detector HOG

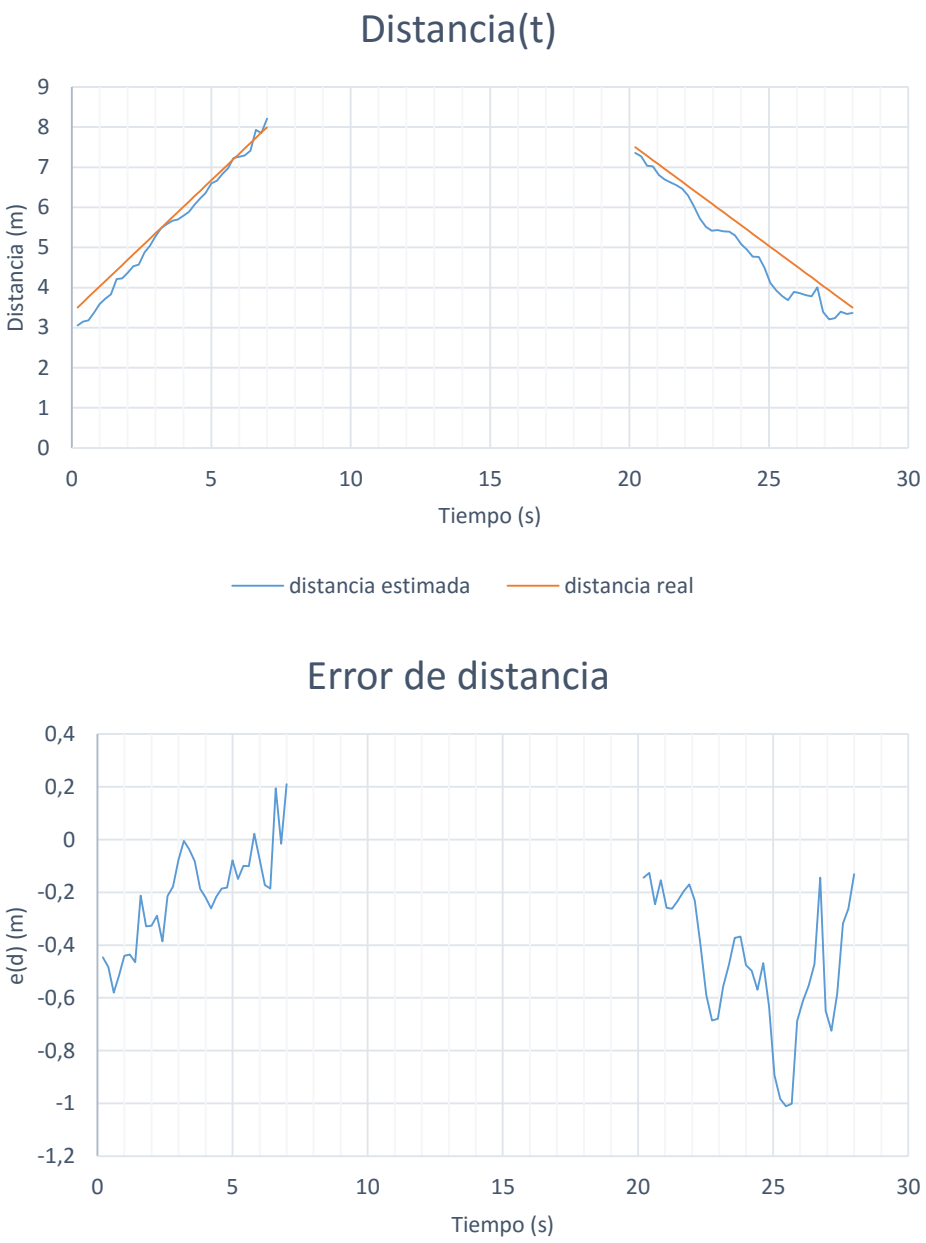


Figura 56, gráfica 11: Trayectoria diagonal, seguida en coordenadas reales, utilizando el detector HOG

En todos los casos analizados se puede apreciar que la trayectoria seguida por la aplicación no se ajusta exactamente a la real. Este error de seguimiento tiene básicamente cuatro fuentes claramente identificadas. La primera fuente de error consiste en que el sujeto de pruebas se encontrase fuera de la trayectoria ideal. En principio las pruebas se han realizado con el cuidado necesario para minimizar en lo posible este problema. La segunda fuente de error se debe a la variación excesiva del rectángulo envolvente al peatón en dos frames consecutivos. Como ya se ha comentado, este es un error muy común en ambos detectores utilizados. En tercer lugar existe un error asociado al filtro de Kalman. Este error se produce porque el movimiento de un peatón no es de velocidad constante, se producen aceleraciones durante su trayectoria. El filtro de Kalman utilizado es un filtro lineal, con lo que en principio el error de seguimiento podría ser

grande. No obstante, si contamos con una tasa de imágenes por segundo relativamente alta y sobre todo, si somos capaces de modelar correctamente la matriz de covarianzas del error de modelo (Q), podemos minimizar en gran medida esta fuente de error. En el apartado 4.6 se ofrecen más detalles al respecto. Por último existe un error asociado al sistema de estimación de distancias. Este error a su vez puede tener varias fuentes. Es posible que el suelo no sea completamente plano (lo cual es necesario, consultar el apartado 4.8), puede haber un error en la medida extraída del acelerómetro del móvil, o puede deberse a la distorsión de la lente (esto se explica con más detalle más adelante en este mismo apartado).

En la siguiente tabla se muestra una lista más amplia de resultados. En ella se incluye la trayectoria seguida, el ratio de detección, los falsos positivos y los errores medios del sistema de seguimiento. También se indica el detector utilizado en cada caso.

Trayectoria	FRAMES	FN	RATIO	FP	$\bar{e}$	$\overline{ABS[e]}$	e_{RMS}	DETECTOR
Coordenadas de la imagen (U,V)								
x = 0	47	10	78.72	0	0.032	0.088	0.001	CASCADE
x = 0	47	10	78.72	0	-0.064	0.065	0.0057	HOG
x = 3	49	8	83.67	0	-0.215	0.233	0.073	CASCADE
x = 3	44	8	81.82	0	-0.239	0.246	0.075	HOG
x = 5	75	20	73.33	5	0.2056	0.2435	0.1027	CASCADE
x = 5	47	17	63.83	0	-0.13	0.205	0.055	HOG
y = 5	15	1	93.33	0	0.052	0.2542	0.096	HOG
y = 5	28	2	93.1	0	0.54	0.83	1.21	HOG
y = 10	37	18	51.35	0	-1.225	1.376	2.57	HOG
y = 15	42	35	16.17	0	0.7642	0.8087	0.75	HOG
Diagonal1	29	4	86.21	0	-0.491	0.4952	0.3331	HOG
Diagonal2	33	7	78.79	0	-0.645	0.669	0.5654	HOG
Trayectoria	**FRAMES**	**FN**	**RATIO**	**FP**	$\bar{e}$	$\overline{ABS[e]}$	e_{RMS}	**DETECTOR**
Coordenadas reales (X,Y)								
x = 0	64	16	75	0	-0.354	0.354	0.1354	CASCADE
x = 0	52	12	76.92	0	-0.442	0.442	0.22	HOG
x = 3	41	5	87.8	0	-0.322	0.322	0.1418	HOG
x = 3	54	14	74.07	0	-0.329	0.329	0.1467	HOG
x = 5	41	11	73.17	0	-0.208	0.2492	0.078	HOG
x = 5	61	18	70.49	0	0.063	0.4	0.244	HOG
y = 5	19	0	100	0	0.4872	0.7731	1.162	HOG
y = 5	28	0	100	0	-0.22	0.292	0.1436	HOG
y = 10	28	26	31.58	0	-2.4	2.4	5.82	HOG

y = 15	26	22	15.38	0	-3.9	3.98	18.3	HOG
Diagonal1	44	17	61.36	0	-0.536	0.5917	0.7763	HOG
Diagonal2	34	9	73.53	0	-0.819	0.8816	1.098	HOG

Tabla 3: Resumen detallado de los resultados obtenidos para distintas configuraciones

En la tabla superior FN significa falsos negativos. Son todos aquellos frames en los que el detector no ha encontrado un objeto que en realidad si existía. Ratio hace referencia al ratio de detección, es decir, el cociente entre verdaderos positivos y el número total de frames en los que aparecen objetos a detectar. FP se refiere a los falsos positivos; todos aquellos frames en los que el detector ha señalado como persona algo que en realidad no lo era. Por último, antes del detector, se indica para cada configuración el error medio del sistema de tracking, el error absoluto medio y el error medio cuadrático.

A la vista de los resultados obtenidos, se pueden destacar las siguientes conclusiones:

- El hecho de que los resultados dependan de los factores limitantes, tanto internos como externos, mencionados en apartados anteriores hace que sea complicado extraer unas conclusiones claras a partir de las pruebas efectuadas en aplicaciones de este tipo. Todo depende del entorno en el que se realicen.

- El punto débil de este sistema, así como de la mayoría de sistemas similares, es el detector [44]. En este caso en concreto no solo afecta el hecho de que el porcentaje de detección no sea excesivamente alto, también afecta en gran medida el hecho de que el rectángulo envolvente asociado al objeto varíe. Si esta variación es grande, la variación en distancia también lo es, lo que se puede traducir en un error de estimación del filtro de Kalman y de seguimiento. De hecho esta variación del rectángulo envolvente supone la mayor fuente de error de la aplicación.

- La lente de la cámara de un smartphone presenta distorsión, la cual es inapreciable en el centro de la imagen, pero aumenta a medida que nos acercamos al borde de la misma. Este fenómeno consiste en que las líneas que son rectas en la realidad se visualizan en la imagen como líneas curvas. Esta distorsión provoca que la coordenada 'v' o vertical de un determinado objeto situado cerca del borde de la pantalla no sea la real, lo que provoca un error en la distancia estimada. La distorsión apreciada en las imágenes captadas por el móvil es la llamada distorsión de cojín.

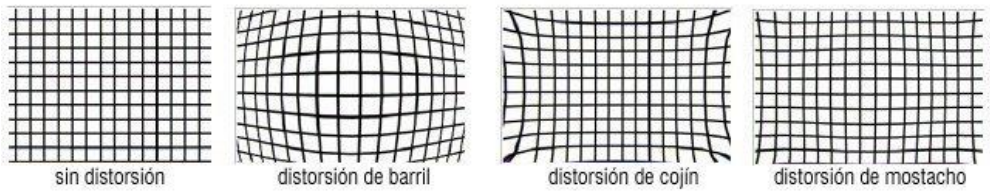


Figura 57: distintos tipos de distorsión de la lente [72]

- Se puede observar como el filtro de Kalman es capaz, en general, de suavizar la trayectoria del objeto seguido. Existen algunas excepciones producidas por un

cambio repentino considerablemente grande en el tamaño del recuadro que envuelve al objeto. Como ya se ha explicado esto es debido al detector.

- La aplicación desarrollada apenas presenta falsos positivos, pues del total de pruebas realizadas solo se han detectado 5. Esto se debe a que la aplicación está configurada para que valide un objeto solo si este aparece al menos en dos frames consecutivos.

Figura 58: ejemplo de falso positivo

- La aplicación comienza a detectar peatones cuando estos se encuentran a una distancia de unos 3.5 metros, ya que esta es la distancia a la cual el peatón se muestra en su totalidad en la imagen. Por otra parte, la detección es satisfactoria hasta una distancia de unos 12 metros hacia delante, a partir de este momento la detección es muy baja, llegando a detectar peatones a unos 14 metros en contadas ocasiones. En el caso del detector Haar Cascade esto podría mejorar disminuyendo el tamaño mínimo de los objetos a buscar. No obstante esta solución conlleva una disminución considerable de la tasa de imágenes por segundo y aumenta mucho el número de falsos positivos de la aplicación. También hay que mencionar que los archivos de entrenamiento utilizados en HOG y Haar Cascade solo permiten detectar a una persona si esta se muestra por completo en la imagen. No puede haber una parte del cuerpo fuera de la pantalla.

- Uno de los objetivos principales de estas pruebas es el de seleccionar de entre todas las configuraciones posibles la que mejor resultados ofrezca. En cuanto al detector, sin duda el ganador es HOG, pues presenta un ratio de detección mayor y permite detectar personas que se encuentren de perfil a la cámara, razón por la cual se ha preferido este último detector al realizar pruebas en trayectorias de coordenada ‘y’ constante. El filtro de Kalman utilizado es lineal, mientras que las ecuaciones utilizadas en el cálculo de distancias son no lineales, con lo que utilizar coordenadas reales para el seguimiento afecta al rendimiento del filtro. Por tanto la configuración elegida es el detector HOG junto con un sistema de coordenadas de la imagen para el seguimiento de los objetos.

Para finalizar el apartado de resultados, se muestra una tabla en la que se puede observar el tiempo de procesamiento que requiere cada una de las etapas en las que se divide la aplicación. Los valores numéricos indicados en la tabla están expresados en milisegundos.

DETECTOR	E0	E1	E2	E3	E4	E5	E6	E7	E8
SIN PEATONES									
CASCADE	0.017	20	0.039	0.011	0.061	0.081	0.041	0.013	0.34
HOG	0.020	165	0.055	0.058	0.11	0.086	0.024	0.011	0.38
CON 3 PEATONES									
CASCADE	0.076	25	0.040	0.17	0.17	0.25	0.34	0.012	2.16
HOG	0.020	160	0.040	0.018	0.29	0.25	0.25	0.16	1.20
SIN COCHES									
CASCADE	0.049	59	0.038	0.011	0.051	0.077	0.022	0.014	0.26
CON 3 COCHES									
CASCADE	0.023	90	0.037	0.15	0.13	0.22	0.26	0.15	1.06

Tabla 4: Tiempo de procesamiento de las diferentes etapas de la aplicación

En la tabla superior se puede ver en primer lugar como el tiempo de procesamiento requerido por el detector HOG para analizar la imagen es muy superior al requerido por Haar Cascade (E1), aun cuando la resolución a la que trabaja HOG es menor. La etapa 4 corresponde a la estimación de distancias a los objetos detectados. Es por ello que el tiempo de procesamiento de esta atapa aumenta con el número de objetos detectados. La etapa 5 corresponde a la aplicación del Algoritmo Húngaro, con lo que el tiempo consumido en esta etapa también aumenta considerablemente con el número de objetos detectados y en seguimiento. Las etapas 3, 6 y 7 corresponden a los siguientes procesos del filtro de Kalman: estimación de la posición de los objetos en seguimiento en el frame actual, actualización de su estado y creación de nuevos objetos en seguimiento a partir de nuevas detecciones. El tiempo de procesamiento para estas etapas aumenta por tanto con el número de objetos en seguimiento.

En el mejor de los casos el detector de objetos ocupa el 89 % del tiempo total de procesamiento de cada frame. Por tanto podemos concluir que el factor limitante de la aplicación, en cuanto a rendimiento se refiere, es el detector y no los demás procesos, de manera casi independiente al número de objetos en seguimiento.

6. TRABAJOS FUTUROS

En este apartado se detallan las posibles mejoras que se pueden aplicar al proyecto aquí desarrollado. Estas mejoras podrían ser desarrolladas por alumnos de la universidad como parte de su TFG en años venideros.

- Desarrollo de un sistema de alarma al conductor

Una de las opciones de mejora más evidentes que presenta la aplicación consiste en desarrollar un sistema de alarma que avise al conductor de la posibilidad de sufrir un accidente. Al final este es el objetivo de cualquier aplicación que pertenezca al campo de los ADAS. El procedimiento consistiría en utilizar el filtro de Kalman para predecir la posición del peatón o coche en frames futuros. Comparando esta información con la trayectoria actual del vehículo se podría deducir la posibilidad de choque o atropello, avisando al conductor de tal situación. La alarma podría consistir en que el móvil hiciese un ruido o que la pantalla empezase a parpadear con colores vivos, o ambas cosas al mismo tiempo.

- Ampliación de la detección a más agentes de la carretera

Sería interesante también incluir en la aplicación la posibilidad de detectar otros agentes de la vía, como pueden ser las bicicletas y las motocicletas. Con ello aumentaríamos la seguridad de la conducción.

Otra opción interesante, relacionada con lo anterior, sería poder unir todas estas detecciones en una sola aplicación. Esto a día de hoy no es demasiado factible debido a que para cada tipo de objeto a detectar se necesita un detector dedicado. Poner a trabajar varios detectores de forma simultánea conlleva un consumo de recursos difícilmente abordable. No obstante, como ya se indicó en la introducción de este trabajo, la tecnología no cesa en su crecimiento, de tal forma que pronto podremos ver dispositivos móviles con la capacidad de cómputo necesaria para afrontar esta tarea.

- Utilización de un entrenamiento propio

Para la detección de peatones y coches se han utilizado tanto los detectores como los archivos de entrenamiento incluidos en OpenCV. Sería deseable poder contar con nuestro propio entrenamiento, pues no sabemos a ciencia cierta cómo se han generado los que estamos utilizando, ni su fiabilidad exacta. Además seríamos capaces de controlar más a fondo todos los aspectos de la aplicación, siendo capaces de mejorar con ello la misma. Podríamos añadir más información y reentrenar a los detectores cuando quisiésemos. Tendríamos también un mayor conocimiento de los puntos débiles y fuertes que presenta nuestro sistema de detección.

- Utilización de otros algoritmos de detección existentes

Los algoritmos utilizados en el desarrollo de la aplicación no son los mejores que existen, por tanto, explorar otras alternativas podría mejorar la detección de todos los agentes de la vía

necesarios. Según se indica en [44] los tres mejores algoritmos de detección para peatones son MULTIFTR+MOTION [46], CHNFTRS [19], y FPDW [45]. El primer detector es significativamente más lento que HOG y Haar Cascade, por lo que su implantación seguramente no sería posible. Sin embargo, CHNFTRS y FPDW son más rápidos que los dos algoritmos aquí utilizados. Estos serían por tanto unos buenos candidatos.

FPDW está basado en histogramas de gradientes orientados, pero utiliza una técnica de aproximación para disminuir el tiempo necesario para crear y analizar la pirámide de imágenes que ha de generarse para cada frame en el que se buscan objetos. Gracias a esta técnica piramidal los detectores son capaces de encontrar objetos de distintos tamaños. CHNFTRS por su parte genera una serie de canales a partir de una sola imagen aplicando sobre ella transformaciones lineales y no lineales. Después extrae características de cada uno de estos canales y las utiliza para entrenar un clasificador Adaboost.

- Utilizar otras características de los objetos a detectar

Junto con estos algoritmos de detección se podrían utilizar otras características tales como el color para mejorar el funcionamiento de la aplicación [47]. Esta característica en concreto resultaría especialmente beneficiosa para el sistema de seguimiento. Calculando un histograma de cada persona detectada se facilita la tarea de tener que asignar las nuevas detecciones a objetos ya existentes en anteriores frames, a cambio de un mayor coste computacional. Su efectividad en el caso de vehículos sería similar.

Una vez concluidas todas estas mejoras, o incluso otras más no expuestas aquí, la aplicación estaría preparada para utilizarse como parte de un sistema de conducción asistida o incluso autónoma. Por ejemplo, se podría utilizar en el coche IVVI 2.0 de la Universidad Carlos III.

7. PRESUPUESTO

En este apartado se muestran los costes de personal y de materiales que ha generado el presente proyecto durante su ejecución.

Para el cálculo de los costes materiales hay que incluir el uso de un ordenador portátil Asus X54H, un teléfono móvil BQ Aquaris E5 4G y un móvil Sony Xperia J utilizado en las primeras etapas de desarrollo del proyecto. Los costes de licencia de software son cero, ya que tanto OpenCV como la librería JKalman son Open Source. Android Studio y Eclipse pueden ser utilizados gratuitamente y el código modificado del Algoritmo Húngaro no tiene derechos de autor.

Para el cálculo del coste de cada elemento material se ha tenido en cuenta el coeficiente de amortización aplicable en cada caso así como el tiempo durante el cual se ha utilizado dicho elemento. Se ha utilizado para ello la tabla de amortización simplificada del Ministerio de Hacienda [67]. La fórmula para realizar el cálculo es la siguiente:

$$Coste = precio \cdot \frac{coef.de\ amortización}{100} \cdot \frac{tiempo\ (meses)}{12}$$

Los costes de personal se han calculado en base al sueldo medio de un ingeniero con menos de un año de experiencia profesional en el campo de la investigación [68] durante el año 2015, lo que da un valor de 18.119 €/año brutos.

COSTES DE MATERIAL				
ELEMENTO	**PRECIO (€)**	**COEFICIENTE DE AMORTIZACIÓN**	**TIEMPO UTILIZADO**	**COSTE PARA EL PROYECTO (€)**
Asus X54HY	479,89	26%	6	62,39
BQ Aquaris E5 4G	239	26%	4	20,71
Sony Xperia J	159	26%	2	6,89
			SUBTOTAL	89,99

COSTES DE PERSONAL		
SUELDO BRUTO ANUAL	**DURACIÓN DEL PROYECTO**	**COSTE (€)**
18.119 €	6 meses	9059,5

COSTE TOTAL	**9.149,49 €**

Tabla 5: Cálculo del presupuesto del proyecto

8. CONCLUSIONES

Se puede concluir que la aplicación cumple de forma adecuada con su cometido, ya que es capaz de detectar peatones y coches y de seguirlos con un error siempre inferior a un metro a una distancia de hasta diez metros por delante de la cámara. Sin duda existen algoritmos de detección y seguimiento (como el filtro de Kalman extendido o el UKF) capaces de ofrecer unos resultados superiores. Mo obstante, la capacidad de cálculo de los teléfonos inteligentes de hoy en día no es lo suficientemente grande como para ejecutar este tipo de algoritmos en tiempo real de forma adecuada.

A lo largo del desarrollo de la aplicación se ha probado el funcionamiento de la misma para cuatro configuraciones diferentes, utilizando para ello dos detectores de objetos distintos, Haar Cascade y HOG, y utilizando un sistema de coordenadas basado en coordenadas de la imagen y otro basado en coordenadas reales. Tras las pruebas realizadas se ha determinado que el mejor funcionamiento se consigue utilizando el detector HOG junto con un sistema de coordenadas de la imagen para el apartado de seguimiento. No obstante, es necesario contar con un procesador rápido en el dispositivo para poder ejecutar la aplicación en tiempo real.

Una de las mayores dificultades encontradas durante el desarrollo del proyecto consiste en la universalización de los parámetros óptimos de funcionamiento. Los sistemas comerciales similares se ejecutan sobre software y hardware dedicado, lo que implica que siempre presentan el mismo rendimiento ante condiciones externas similares. En este caso el problema reside que la aplicación debe funcionar de manera similar para cualquier dispositivo móvil. Cada dispositivo tiene unas características de hardware únicas, lo que dificulta esta tarea. La aplicación se ha desarrollado de manera que se pueda adaptar fácilmente, y en muchos casos de forma automática, ante cambios en la velocidad de procesamiento y resolución. Sin embargo, no se puede garantizar que el rendimiento sea igual en distintos dispositivos.

Una vez que se implementen las funciones adicionales descritas en el apartado de trabajos futuros, y teniendo a nuestra disposición dispositivos móviles ligeramente más rápidos en capacidad de cálculo, estaríamos ante una aplicación totalmente aplicable a un entorno real, totalmente funcional y, lo más importante, portable y capaz de ejecutarse en cualquier dispositivo móvil que ejecute el sistema operativo Android.

9. REFERENCIAS

[1] Informe de la OMS en 2015 sobre la seguridad del tráfico por carretera: http://www.who.int/violence_injury_prevention/road_safety_status/2015/en/ [Último acceso: noviembre 2015]

[2] Informe de Morgan & Stanley Nikola's Revenge: TSLA's New Path of Disruption, Tesla Motors Inc., Morgan Stanley, Feb.25 2014.

[3] Mobileye 5 product series. http://www.mobileye.com/products/mobileye-5-series/ [Último acceso: noviembre 2015]

[4] Asistente de ángulo muerto de Mercedes-Benz. http://techcenter.mercedes-benz.com/es_ES/blind_spot_assist/detail.html [Último acceso: noviembre 2015]

[5] Control activo del ángulo muerto de Mercedes-Benz. http://techcenter.mercedes-benz.com/es_ES/active_blind_spot_assist/detail.html [Último acceso: noviembre 2015]

[6] Audi Side Assist. https://www.audi.co.uk/glossary/a/audi-side-assist.html [Último acceso: noviembre 2015]

[7] Active Parking Assist de Mercedes-Benz. http://techcenter.mercedes-benz.com/en/active_parking/detail.html [Último acceso: noviembre 2015]

[8] Valeo Beep&park/vision. http://www.sonriaestaconduciendo.com/es_es-products-parking-assistance.innovative-solution-3.html [Último acceso: noviembre 2015]

[9] Toyota Lane Keeping Assist. http://www.toyota-global.com/innovation/safety_technology/safety_technology/technology_file/active/lka.html [Último acceso: noviembre 2015]

[10] Lane Departure Warning System. https://en.wikipedia.org/wiki/Lane_departure_warning_system [Último acceso: noviembre 2015]

[11] Sistema de seguridad Opel Eye. http://www.opel.es/vehiculos/coches-opel/vehiculos-de-pasajeros-opel/insignia-4-door/claves/seguridad.html [Último acceso: noviembre 2015]

[12] Sistemas de reconocimiento de señales de tráfico en turismos. Colaboración RACC-ADAC. http://imagenes.w3.racc.es/uploads/file/22207_Sistema_Reconocimiento_Senyales.p

df
[Último acceso: noviembre 2015]

[13] Google Car.
https://www.google.com/selfdrivingcar/
[Último acceso: noviembre 2015]

[14] Volvo Drive-Me.
http://www.volvocars.com/es/acerca-de-volvo/innovaciones/coches-inteligentes/coche-autonomo
[Último acceso: noviembre 2015]

[15] Tesis doctoral de Daniel Olmeda Reino, investigador de la UC3M.
http://earchivo.uc3m.es/bitstream/handle/10016/18665/tesis_daniel_olmeda_2014.pdf?sequence=1
[Último acceso: noviembre 2015]

[16] Un coche inteligente de la UC3M detecta peatones de noche cuadrito. Oficina de información científica de la UC3M.
http://portal.uc3m.es/portal/page/portal/actualidad_cientifica/noticias/coche_inteligente_peatones
[Último acceso: noviembre 2015]

[17] Vehículo robótico Stanley de la Universidad de Stanford.
http://cs.stanford.edu/group/roadrunner//old/index.html
[Último acceso: noviembre 2015]

[18] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. Conference on Computer Vision and Pattern Recognition (CVPR), 2005.

[19] P. Dollar, Z. Tu, P. Perona, and S. Belongie, “Integral channel features,” in BMVC, 2009.

[20] CPU CascadeClassifier with HOG not working anymore with OpenCV 3.0.0.rc1.
https://github.com/Itseez/opencv/issues/4989
[Último acceso: diciembre 2015]

[21] JKalman. http://sourceforge.net/projects/jkalman/
[Último acceso: marzo 2015]

[22] Sensor Overview. Android developers.
http://developer.android.com/intl/es/guide/topics/sensors/sensors_overview.html
[Último acceso: enero 2016]

[23] Artículo de Android en Wikipedia.
https://es.wikipedia.org/wiki/Android
[Último acceso: diciembre 2015]

[24] About Android. Android Developers.
http://developer.android.com/intl/es/about/android.html
[Último acceso: octubre 2015]

[25] «Android-System-Architecture» de Smieh - Anatomy Physiology of an Android. Disponible bajo la licencia CC BY-SA 3.0 vía Wikimedia Commons -
https://commons.wikimedia.org/wiki/File:Android-System-

Architecture.svg#/media/File:Android-System-Architecture.svg
[Último acceso: diciembre 2015]

[26] ART en Wikipedia.
https://es.wikipedia.org/wiki/Android_Runtime_(ART)
[Último acceso: diciembre 2015]

[27] Android Developers. Activities.
http://developer.android.com/intl/es/guide/components/activities.html
[Último acceso: febrero 2016]

[28] OpenCV en itseez.
http://itseez.com/OpenCV/
[Último acceso: octubre 2015]

[29] OpenCV website. http://opencv.org/
[Último acceso: febrero 2016]

[30] Developing OpenCV computer vision apps for the Android platform.
http://www.embedded.com/design/programming-languages-and-tools/4406164/Developing-OpenCV-computer-vision-apps-for-the-Android-platform
[Último acceso: julio 2015]

[31] Android Developers. Android NDK.
http://developer.android.com/intl/es/tools/sdk/ndk/index.html
[Último acceso: noviembre 2015]

[32] Artículo sobre Eclipse en Wikipedia.
https://en.wikipedia.org/wiki/Eclipse_(software)
[Último acceso: diciembre 2015]

[33] Android Developers. Android Studio and SDK Tools.
http://developer.android.com/intl/es/sdk/index.html
[Último acceso: noviembre 2015]

[34] Android Developers. Android Studio Overview.
http://developer.android.com/intl/es/tools/studio/index.html
[Último acceso: enero 2016]

[35] D. Martín, F. García, B. Musleh, D. Olmeda, G. Peláez, P. Marín, A. Ponz, C. Rodríguez, A. Al-Kaff, A. de la Escalera, J.M. Armingol, IVVI 2.0: An intelligent vehicle based on computational perception, Expert Systems with Applications, Volume 41, Issue 17, 1 December 2014, Pages 7927-7944, ISSN 0957-4174,
http://dx.doi.org/10.1016/j.eswa.2014.07.002.
[Último acceso: febrero 2016]

[36] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. Conference on Computer Vision and Pattern Recognition (CVPR), 2001.

[37] Cascade Classification.
http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html
[Último acceso: noviembre 2015]

[38] Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software. Yaakov Bar-Shalom, X. Rong Li, Thiagalingam Kirubarajan

[39] Kalman filter. Wikipedia.
https://en.wikipedia.org/wiki/Kalman_filter
[Último acceso: enero 2016]

[40] Hungarian Method.
http://www.math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf
[Último acceso: enero 2016]

[41] Arturo de la Escalera Hueso, "Visión por Computador, Fundamentos y Métodos", Prentice Hall, Madrid, 2001.

[42] A C implementation of the Hungarian Method.
http://robotics.stanford.edu/~gerkey/tools/hungarian.html
[Último acceso: noviembre 2015]

[43] Munkres' Assignment Algorithm. Modified for Rectangular Matrices.
http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html
[Último acceso: noviembre 2015]

[44] P. Dollár, C. Wojek, B. Schiele and P. Perona. Pedestrian Detection: An Evaluation of the State of the Art. PAMI, 2012.
http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/files/PAMI12pedestrians.pdf
[Último acceso: diciembre 2015]

[45] P. Dollar, S. Belongie, and P. Perona, "The fastest pedestrian detector in the west," in British Machine Vision Conf., 2010.

[46] S. Walk, N. Majer, K. Schindler, and B. Schiele, "New features and insights for pedestrian detection," in IEEE Conf. Computer Vision and Pattern Recognition, 2010.

[47] Z. Jiang, D. Q. Huynh, W. Moran, S. Challa, N. Spadaccini. Multiple Pedestrian Tracking using Colour and Motion Models. In first Int. Conference on Digital Image Computing: Techniques and Applications (DICTA'01), proceedings, Sydney-Australia, December 2010.

[48] iOnRoad Augmented Driving Android app in Google Play.
https://play.google.com/store/apps/details?id=com.picitup.iOnRoad.pro&hl=es
[Último acceso: febrero 2016]

[49] Road Sign Recognition app in Google Play.
https://play.google.com/store/apps/details?id=road.signs.recognition&hl=es
[Último acceso: febrero 2016]

[50] myDriveAssist app in Google Play.
https://play.google.com/store/apps/details?id=com.bosch.mydriveassist&hl=es
[Último acceso: febrero 2016]

[51] Mobileye 5 - Series pro app in Google Play.
https://play.google.com/store/apps/details?id=com.mobileye.medisplay&hl=es
[Último acceso: febrero 2016]

[52] aCoDriver 4 app in Google Play.
https://play.google.com/store/apps/details?id=com.evotegra.aCoDriver&hl=es
[Último acceso: febrero 2016]

[53] Ángulo muerto: cómo evitar accidentes al conducir por las calles.
http://www.todoautos.com.pe/portal/auto/seguridad/1649-angulo-muerto-como-evitar-accidentes-al-conducir-por-las-calles
[Último acceso: febrero 2016]

[54] VERSO CORNERING ASSIST (2007 – 2009)
http://media.toyota.co.uk/image_library/verso-cornering-assist-2007-2009/
[Último acceso: febrero 2016]

[55] Système d'aide à la conduit
http://www.trw.fr/integrated_systems/driver_assist_systems
[Último acceso: febrero 2016]

[56] Artículo web de la página Xataka: Para 2020 tendremos coches que evitarán todo tipo de accidentes, según Volvo.
http://www.xataka.com/automovil/para-2020-tendremos-coches-que-evitaran-todo-tipo-de-accidentes-segun-volvo
[Último acceso: febrero 2016]

[57] Collision warning – Pedestrian detection, Volvo.
http://support.volvocars.com/en-CA/cars/Pages/owners-manual.aspx?mc=y286&my=2016&sw=15w17&article=d3d274ecedf1c586c0a801e8004927e7
[Último acceso: febrero 2016]

[58] Volvo City Safety.
http://www.volvocars.com/es/footer/terminos-y-condiciones/city-safety#
[Último acceso: febrero 2016]

[59] Honda Develops World's First Intelligent Night Vision System Able to Detect Pedestrians and Provide Driver Cautions, Honda, August 2004.
http://world.honda.com/news/2004/4040824_01.html
[Último acceso: febrero 2016]

[60] Mercedes se pone las pilas, blog de la página blogs.km77, diciembre 2008.
http://blogs.km77.com/seguramente/176/mercedes-se-pone-las-pilas-i/
[Último acceso: febrero 2016]

[61] Consumers not quite ready for driverless cars, Mary Gannon, June 2013.
http://www.connectortips.com/consumers-not-quite-ready-for-driverless-cars/
[Último acceso: febrero 2016]

[62] Grand Challenge NQE update.
http://rossums-children.blogspot.com.es/2005_09_25_archive.html
[Último acceso: febrero 2016]

[63] Sony Xperia J coming to Bell on March 27th.
http://mobilesyrup.com/2013/03/14/sony-xperia-j-coming-to-bell-on-march-27th/
[Último acceso: febrero 2016]

[64] Aquaris E5 4G.
http://www.bq.com/es/aquaris-e5-4g
[Último acceso: febrero 2016]

[65] Hiding the status bar. Android developers.
http://developer.android.com/intl/es/training/system-ui/status.html
[Último acceso: febrero 2016]

[66] Hiding the Navigation Bar. Android developers.
http://developer.android.com/intl/es/training/system-ui/navigation.html
[Último acceso: febrero 2016]

[67] Tabla de amortización simplificada. Agencia Tributaria.
http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresarios_individuales_y_profesionales/Rendimientos_de_actividades_economicas_en_el_IRPF/Regimenes_para_determinar_el_rendimiento_de_las_actividades_economicas/Estimacion_Directa_Simplificada.shtml
[Último acceso: febrero 2016]

[68] Encuesta de Salarios y Actividad profesional 2014 – 2015. Colegios oficiales de ingenieros industriales de Álava, Vizcaya, Guipúzcoa y Navarra.
http://www.coiig.com/COIIG/dmdocuments/Empleo%20LANBIDE/Salarios/encuesta_salarios_ingenieros_industriales_capv-navarra_2014-2015.pdf
[Último acceso: febrero 2016]

[69] Tesla's Cars Now Drive Themselves, Kinda. Molly Mchugh. October 2015.
http://www.wired.com/2015/10/tesla-self-driving-over-air-update-live/
[Último acceso: febrero 2016]

[70] Biblioteca para localización de peatones en dispositivos Android. Víctor Cancho Armesto, Grado en Ingeniería Electrónica Industrial y Automática. 2014.

[71] Implementación de Algoritmos de Visión por Computador en Plataforma Android. Alejandro Ramos López, Grado en Ingeniería Informática – UC3M. 2013.

[72] Distorsión de lente vs Distorsión de la perspectiva.
http://www.xatakafoto.com/guias/distorsion-de-lente-vs-distorsion-de-la-perspectiva
[Último acceso: febrero 2016]